APACHE SLING & FRIENDS TECH MEETUP
2 - 4 SEPTEMBER 2019

New ways of Server Side Rendering with AEM, React and Node JS

Alexander Schmidt, TECLEAD

About me

- Technology enthusiast
- Passionate developer
- Consultant with 9 years of experience
- Co-Founder of Teclead (https://teclead.de)
- Nature lover (hiking, mountain biking, traveling)
- Enjoy digital goods and of course coding

Agenda

1. Introduction to setup
2. SSR vs CSR
3. SSR approaches
    a) Mozilla Rhino
    b) Custom Node JS Service
    c) Tessellate
    d) Next JS
4. Take away

1. Introduction to setup

adaptTo()

1. Introduction to setup (1/3)

- Greenfield with AEM 6.4 with Touch UI
- Mostly React for AEM components
- Some components with default AEM tool stack
- Gradle - Cognifide for AEM packaging

1. Introduction to setup (2/3)

- Node JS, React, Less, Typescript
- E2E with Webmate
- Webpack to build chunks
- Jenkins for automation (build, deploy, release and delivery)

1. Introduction to setup (3/3)

Why did we choose React?

- already in use and mostly known in company
- rich library of elements already implemented (checkbox, label, accordion, icons, etc.)
- elements were used by other SPA's
- reuse all of that components in AEM
- reduce time and effort
- fast delivery

adaptTo()

2. SSR vs CSR

adaptTo()

SSR vs CSR (1/4)

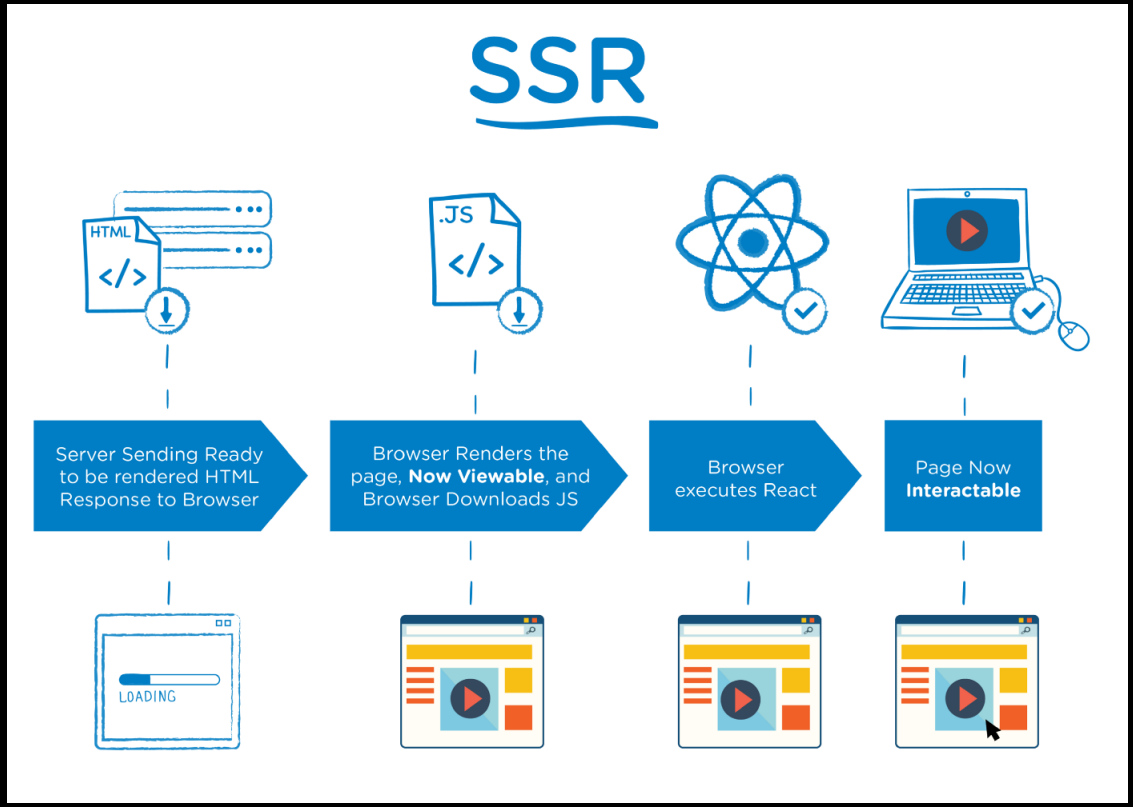
	SSR	CSR
PRO	• better for SEO (more consistent) • reduce load on client side • blank page flicker that happens with CSR, doesn't really happen with SSR	• rendition on client side reduces load on server • better user experience while navigating through pages • load page only once and fetch new data as needed
CON	• rendition on server increase load on server • annoying page loads when server is on heavy load	• not optimal for SEO • load on client side may be dissatisfying with low bandwidth • JavaScript may slow down the page performance

SSR vs CSR (2/4)

walmart.com rendered with SSR vs CSR

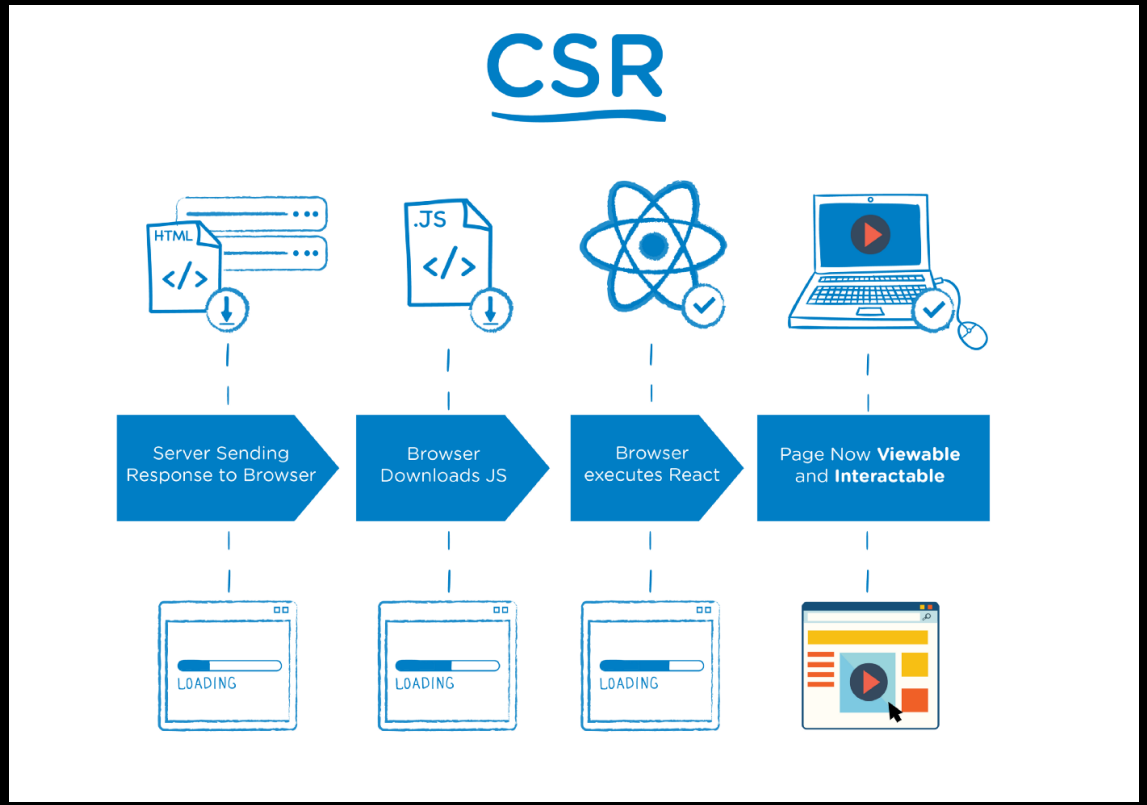
*https://medium.com/walmartlabs/the-benefits-of-server-side-rendering-over-client-side-rendering-5d07ff2cefe8

SSR vs CSR (3/4)



*https://medium.com/walmartlabs/the-benefits-of-server-side-rendering-over-client-side-rendering-5d07ff2cefe8

SSR vs CSR (4/4)



*https://medium.com/walmartlabs/the-benefits-of-server-side-rendering-over-client-side-rendering-5d07ff2cefe8

adaptTo()

3. SSR approaches

3. SSR approaches – introduction (1/3)

- AEM component with HTL template
- React template to render component
- Properties will be injected to React component
- Web bundles stored in JCR

3. SSR approaches – introduction (2/3)

HTL template

```
<!-- simplified -->
<sly data-sly-use.ssr="com.aem.sample.components.ServerSideRenderingComponent">
    <div class="aem-react-component SampleReactComponent"  data-params="${ssr.allProperties}">
        <!-- Rendered content -->
        ${ssr.renderedHtml}
    </div>
</sly>
```

3. SSR approaches – introduction (3/3)

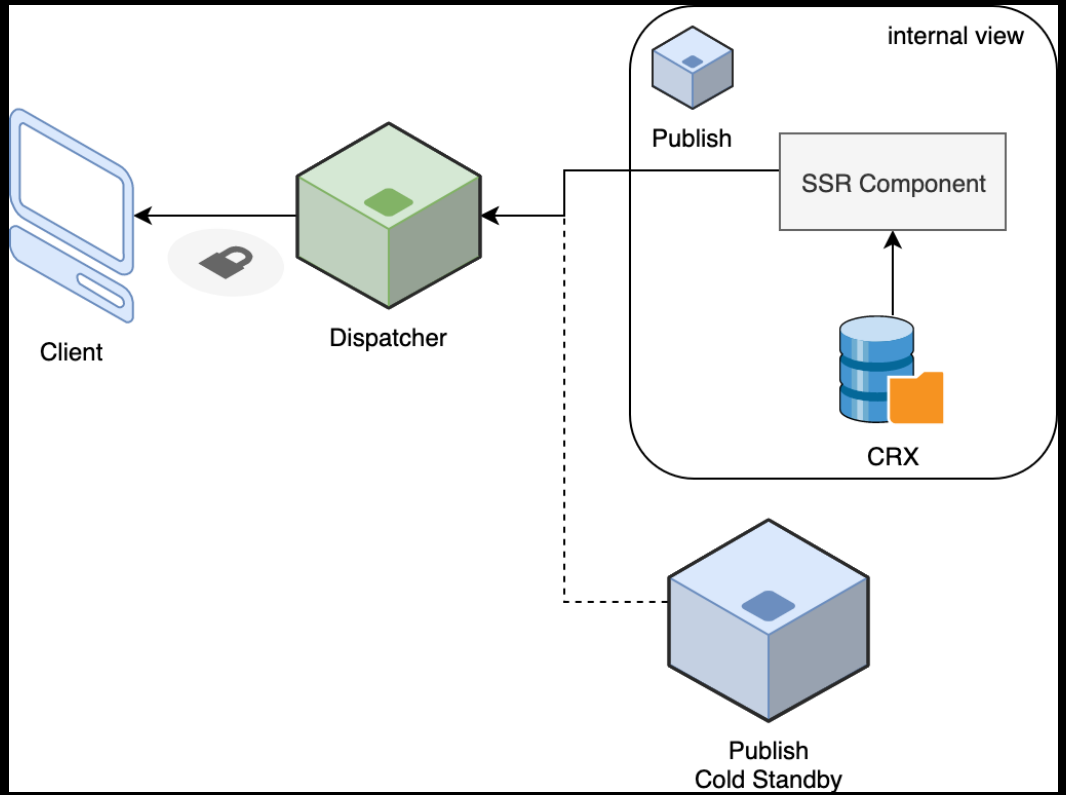
React template

```
import React from 'react';
import ReactDOMServer from 'react-dom/server';
import SampleReactComponent from './SampleReactComponent;

// properties is injected from AEM
let props= properties;
const json = JSON.parse(props);

renderedHtml = ReactDOMServer.renderToString(<SampleReactComponent{...json}></SampleReactComponent>);
```

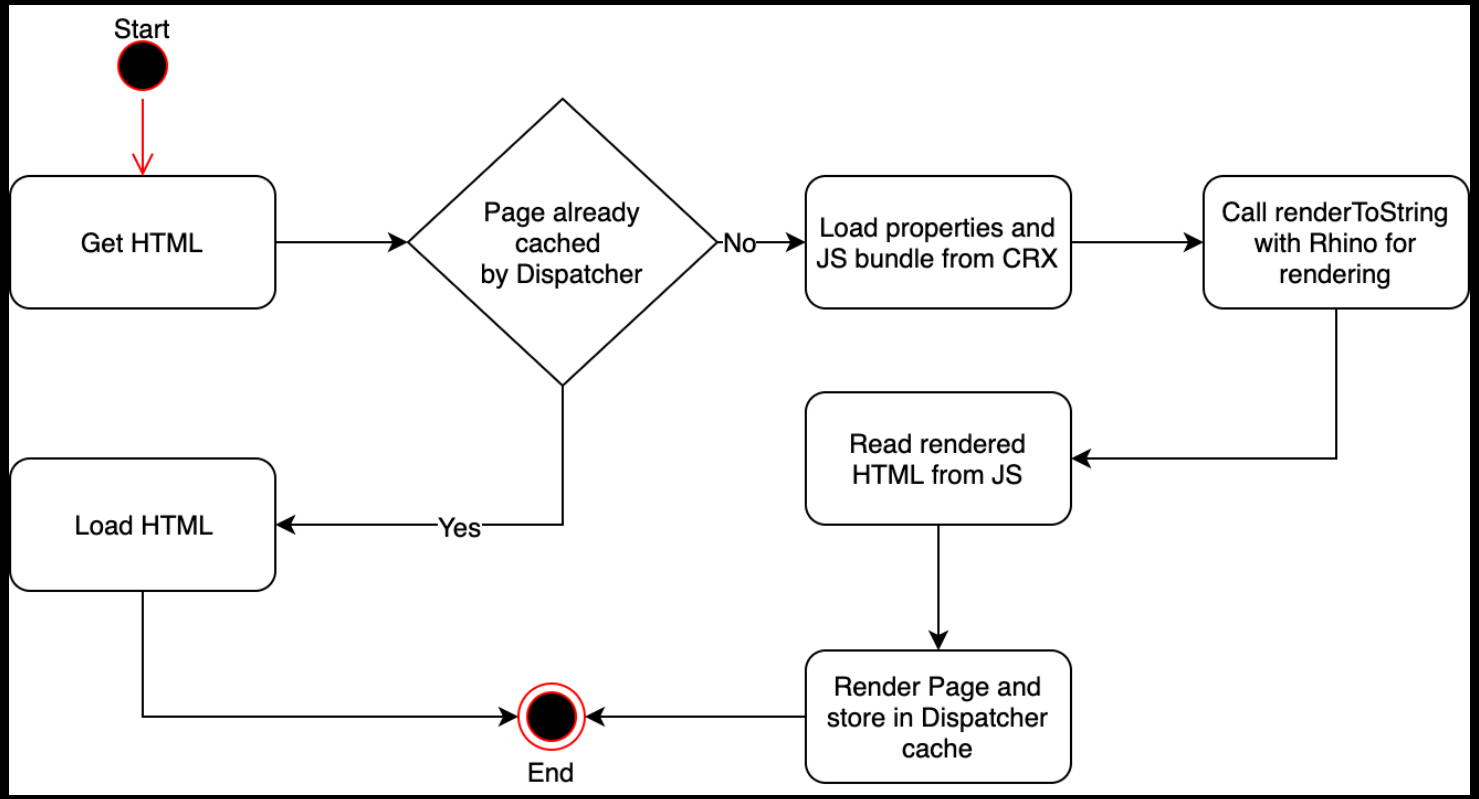
adaptTo()

3a. SSR approaches - Mozilla Rhino

3a. Mozilla Rhino (1/5)



3a. Mozilla Rhino (2/5)



3a. Mozilla Rhino (3/5)

```
public String getRenderedHtml(String bundleContent, String properties) {
    // start JS-Context for evaluation
    Context context = Context.enter();
    ScriptableObject globalScope = context.initSafeStandardObjects();

    // pass props from AEM into the context as global variable
    ScriptableObject.putProperty(globalScope, "properties", Context.javaToJS(properties, globalScope));

    // execute the bundle and access the renderedHtml global variable which
    // contains the html-string of the component
    context.evaluateString(globalScope, bundleContent, JS_BUNDLE_FILE_NAME, 0, null);
    return (String) Context.jsToJava(ScriptableObject.getProperty(globalScope, "renderedHtml"), String.class);
}
```

3a. Mozilla Rhino (4/5)

Advantages

- Can be hosted internally in AEM
- Simple function call
- Everything in one place

3a. Mozilla Rhino (5/5)

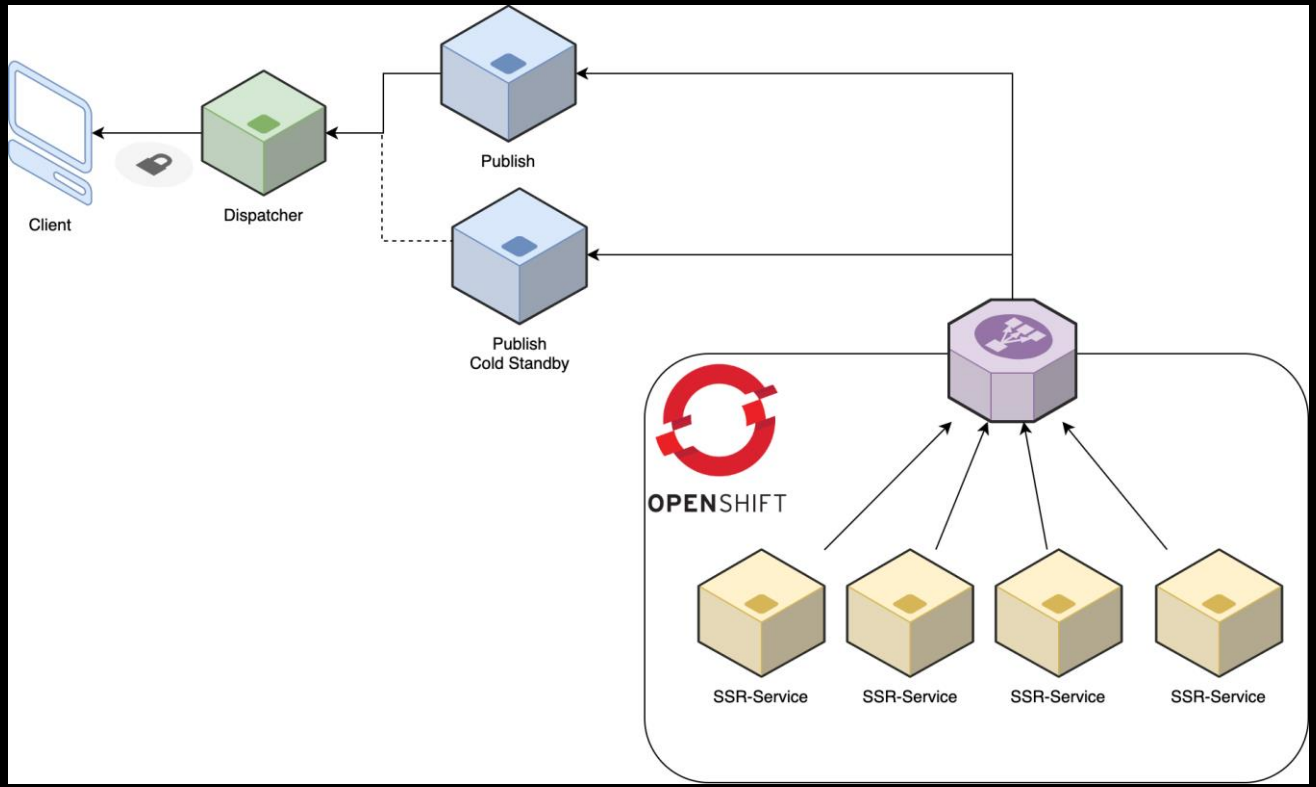
Disadvantages

- Mozilla Rhino is not well maintained
- Javascript functions unknown by Rhino
- Unknown functions have to be implemented
- Unexpected behavior in AEM

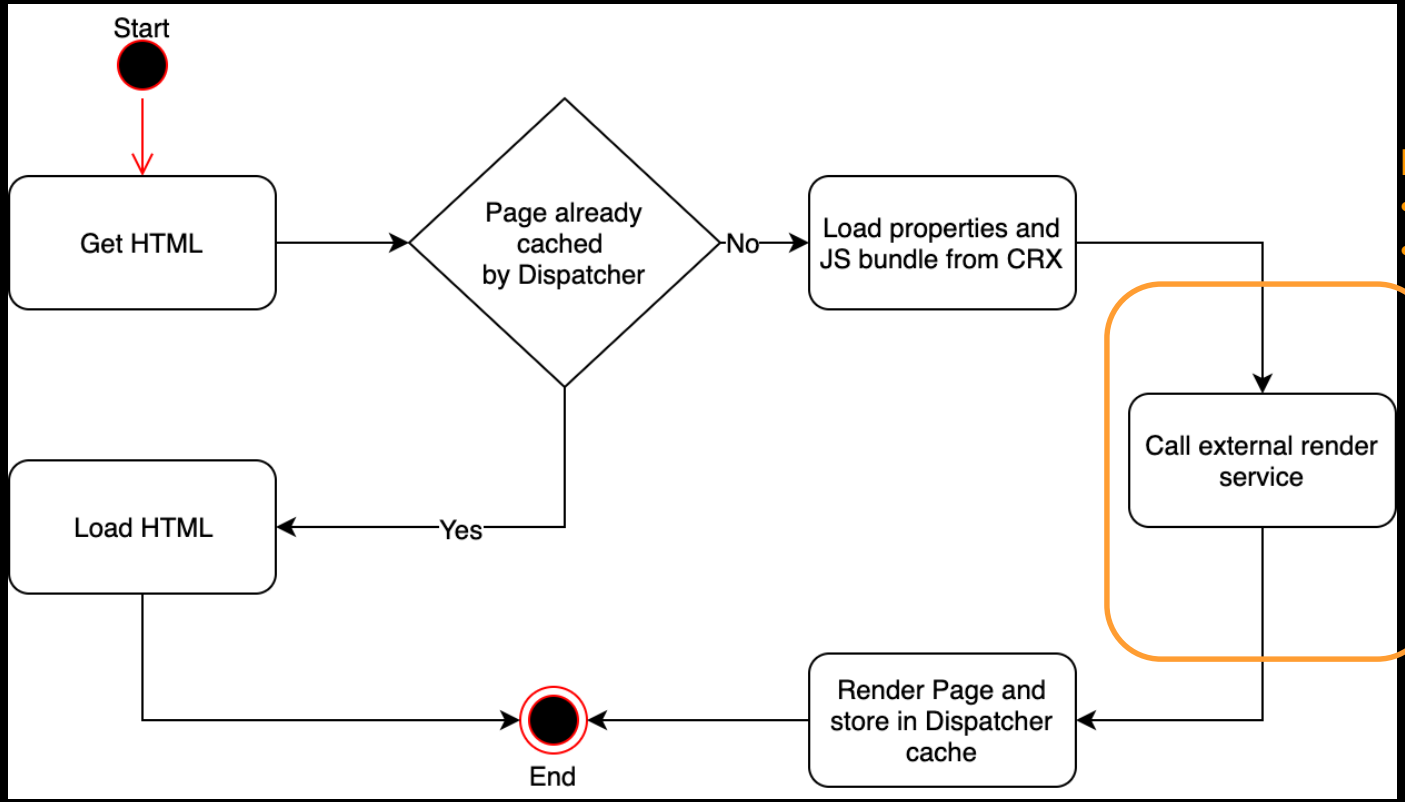
adaptTo()

3b. SSR approaches - Custom Node JS Service

Custom Node JS Service (1/5)



Custom Node JS Service (2/5)



Differences:

- Call a Service
- Read HTML disappears

Custom Node JS Service (3/5)

```
public render(bundleContent: string, props: string): string {
    // init variables
    const vars = {
      properties: props,
      renderedHtml: '',
      window: dom.window,
      document: dom.window.document,
      navigator: dom.window.navigator,
      console: dom.window.console
    };

    // prepare node vm context
    const context = vm.createContext(vars);
    const script = new vm.Script(bundleContent);

    // execute script
    script.runInContext(context);
    return vars.renderedHtml;
}
```

Custom Node JS Service (4/5)

Advantages

- Node JS engine is well maintained
- Load runs on external service
- Service can be reused for other applications
- Better scaling and failure-safety
- Implementation is simple, because of less polyfill

Custom Node JS Service (5/5)

Disadvantages

- More expensive because of external hosting
- HTTP call takes longer than function call

3c. SSR approaches - Tessellate

Tessellate (1/2)

- Developed by Zalando and is open source
- Creates static HTML and JS bundle from JSON definitions
- Webpack bundler and rendering service

Tessellate (2/2)

- Works similar as our custom implementation with Node JS
- Has more features (packages) if needed, but could be unnecessary

3d. SSR approaches - Next JS

Next JS (1/2)

- A React framework for SSR apps and more
- Open source and can be found on Github
- Supports SSR with React out of the box

Next JS (2/2)

- Feature rich
  - Dynamic SSR with service
  - Static HTML generator
  - Desktop, mobile, PWA
- Big community and under further development

adaptTo()

4. Take away

Take away (1/5)

Comparison	Rhino	Node JS Service	Tessellate	Next JS
Reusability				
Complexity				
Costs				
Flexibility				
Maintenance				

* Good | Okay | Bad

adaptTo()

Take away (2/5)

- SSR with Rhino can be very burdensome, because of limited JS engine
- SSR with Node JS works better, because the engine is better maintained
- External hosting of Service is more expensive

Take away (3/5)

- Be aware of features which do not support SSR, e.g. React Portals
- Introducing a whole new technology stack is time consuming and not for the faint-hearted
- Add SSR to your definition of done

Take away (4/5)

- Interdisciplinary teams
- AEM developers for integration
- React developers could implement components

Take away (5/5)

What should you choose?

1. Simple components –> **Rhino**
2. Maximum setting "for yourself"-> **custom Node JS service**
3. Convenience, support and community –> **Tessellate, Next JS**