**Sitecore 9 Certified Platform Associate Developer Exam Prep Quiz**

If you are going for [Sitecore certification exam](https://sitecore.freshdesk.com/support/solutions/articles/16000067853-sitecore-9-0-certified-platform-associate-developer-exam-study-guide-), this exam prep app may help you to pass the exam. It is a micro frontend made by [Preact X](https://github.com/preactjs/preact/releases/tag/10.0.0-alpha.0) and [TypeScript](https://www.typescriptlang.org/) and the bundle is hosted in [GitHub Pages](https://pages.github.com). Source code for this app is [here](https://github.com/mydatahack/sitecore-certification-exam-prep-typescript-preact).

Micro fontends use a microservice approach to front-end. The components on the page can be independent JavaScript apps. Each app is a single bundle. It doesn’t need to use Framework. It can be a vanilla JS that bundled into a single JS file. Let’s discuss the approach I took to build this app.

**Preact**

I used [Preact X](https://github.com/preactjs/preact/releases/tag/10.0.0-alpha.0) with [TypeScript](https://www.typescriptlang.org/) (source code [here](https://github.com/mydatahack/sitecore-certification-exam-prep-typescript-preact)). Preact is a React alternative with much smaller bundle size. Preact X introduced what was lacking in the previous version 8, like Fragment and hooks. I have been using Preact a lot and it just works like React, except the bundle size. If you already know React, no learning curve or surprise to write app in Preact. Unit tests work with the same tools you would use for React.

Minified bundle size for Preact is 9.1kb while React requires 6.5kb (react) + 109.7kb (react-dom). If I use React for the app like the one above would have the bundle size of more than 200kb because react-dom is large. With Preact, the app above is only 70kb. Keeping bundle size small is one of the most important point you need to consider. Preact fits better than React because they have the same functionality, but the former has much smaller bundle size.

**CDN**

The bundle is hosted at the external CDN. In this case, [GitHub Pages](https://pages.github.com) is use because it is free and super easy to use. It can be anything like [S3](https://aws.amazon.com/s3/) if you need a more enterprise -ready solution.

**CSS**

If you want to encapsulate the style into a component, we can do CSS in JS. I didn’t want the component style to clash with the global style available on this site. So, I simply created a Style component and encapsulate css within the bundle. It is safer to prefix all the class names to avoid clash. In this example, [Style.tsx](https://github.com/mydatahack/sitecore-certification-exam-prep-typescript-preact/blob/master/src/components/Style.tsx) is an independent component with returning style element and injected into the React DOM tree. Another popular approach would be to use [styled-component](https://www.styled-components.com) for more complex CSS logic.

**Image**

The image (Sitecore logo) is sourced from a CDN that hosts static assets like images, fonts and icons. Another approach to encapsulate image is to use a component that returns SVG, which works fine as well.

**SEO consideration**

If the components need to be index by search engines, this is not a great approach. You always should go for server-side rendering.

**Putting all together**

After the bundle is ready, all I needed to do was to create a div with an id where the React app can mount. I needed to add some styling to override global CSS that is specific to this site for input label as the global style is applied to the label element.

<script src="https://gist.github.com/mydatahack/323e00833a76cbfa0a0be47b957bbdcf.js"></script>

**Sitecore 9 approach**

In terms of front-end, Sitecore 9 introduced [Sitecore JavaScript Services](https://jss.sitecore.com/) (JSS). With JSS, you can build front-end with whatever JS framework you want to use in a headless manner. Adding React apps in Sitecore prior to 9 wasn’t always smooth. With the introduction of JSS, developers can develop independent front-end components that use Sitecore backend services in a more de-coupled manner. The app gets rendered in the server. So, you don’t need to worry about SEO.

If you want to use this micro frontend approach and leverage Sitecore backend, you can create an API endpoint in Sitecore. Then, micro frontend components can use the endpoint to retrieve or send data. If the page needs to be indexed by search engines, it is not the right solution. Go for JSS or razor view.

Another consideration is site performance. No matter how small the bundle is, the browser still need to get the bundle and render it. If the page has a several micro frontends, it will slow down the page performance significantly.

If the component is simple and used repeatedly throughout the different pages, it may be better to go for [web components](https://developer.mozilla.org/en-US/docs/Web/Web_Components) and create a [custom element](https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_custom_elements). In this way, you can inject the component by simply adding the custom HTML tag wherever it’s needed.

**References**

Here are more useful exam prep materials. Most of the questions in my quiz app are based on them.

**Textbooks**

* [Professional Sitecore 8 Development](https://www.amazon.com/Professional-Sitecore-Development-Solutions-Practices/dp/1484222911)
* [Sitecore Cookbook for Developers](https://www.amazon.com/Sitecore-Cookbook-Developers-Yogesh-Patel/dp/1784396524)

**Online**

* <https://quizlet.com/216770743/flashcards>
* <https://quizlet.com/216770743/flashcards>
* <https://sitecoremaster.weebly.com/sitecore-certification-qa.html>