Programming the mini app way

- What has worked well for mini apps
- Components
- Model-view-viewmodel
- Page-wise thinking
- Build process
- Powerful capabilities
- Acknowledgements

{% Aside %} This post is part of an article collection where each article builds upon previous articles. If you just landed here, you may want to start reading from the beginning. {% endAside %}

What has worked well for mini apps

In this chapter, I want to look at lessons I learned from researching mini apps from a web developer's point of view, or answer the question what does it mean to develop the mini app way.

Components

Rather than reinvent the wheel and make developers build yet another implementation of common UI paradigms like tabs, accordions, carousels, etc., mini apps just ship with a default selection of components that is extensible in case you need more. On the web, there are likewise many options, some of which I have listed in the chapter on mini app components. In an ideal world,

component libraries on the web were built in a way that you could mix them freely. In practice, too many times, there is a certain lock-in regarding a design system you need to buy in to when you use a component, or the component library is distributed in a way that it is all or nothing, but no individual components can be easily added to a project. There are, however, atomic components that you can use in isolation, or libraries like generic-components that are unstyled on purpose. Finding an using those seems like a good idea.

Model-view-viewmodel

The model–view–viewmodel (MVVM) architectural pattern—that facilitates the separation of the development of the graphical user interface (the view) via a markup language from the development of the back-end logic (the model)—means the view is not dependent on any specific model platform. While there are some documented disadvantages of the pattern, in general it works really well for applications of the complexity of mini apps. It can shine especially with rich templating libraries (see next chapter).

Page-wise thinking

Debugging mini apps shows that they are essentially multi-page applications (MPA). This has many advantages, like, for example, it allows for trivial routing and conflict-free per-page styling. People have successfully applied MPA architectures to Progressive Web Apps. Thinking in pages also helps manage resources like each page's CSS and JavaScript files, and other assets like images and videos. Most importantly, building this way means you get route-based code splitting for free if you do not load anything else. In that case, each page by definition strictly only loads what it needs to function.

Build process

Mini apps have no visible build process. On the web, modern build tools like Snowpack leverage JavaScript's built-in module system (known as ESM) to avoid unnecessary work and stay fast no matter how big a project grows. While it is early days for technologies like Web Bundles, it is something that can be easily added to the build process.

Powerful capabilities

The web platform has gained many new capabilities recently. Access to devices via Bluetooth, USB, HID, serial, and NFC is all possible now. Where mini apps run in WebViews and depend on a JavaScript bridge, on the web, these powerful capabilities are available directly, so you do not program against an API provided by the JavaScript bridge, but against the browser API without an intermediate actor.

{% Aside 'success' %} Read on to see an example project that puts this way of programming into practice. {% endAside %}

Acknowledgements

This article was reviewed by Joe Medley, Kayce Basques, Milica Mihajlija, Alan Kent, and Keith Gu.

□□□ #2
□ □□□ □□□ 2026-10-31 22:59:52 UTC
□ □□□ □□□ 2026-10-31 22:59:54 UTC