

Mini Apps

- [index](#)
- [What are mini apps?](#)
- [Other mini app runtime environments](#)
- [Mini app components](#)
- [Concluding thoughts about mini apps from a web developer](#)
- [Mini app DevTools](#)
- [Applying the mini app programming principles to an example project](#)
- [Mini app markup, styling, and scripting](#)
- [Mini app open source projects](#)
- [Programming the mini app way](#)
- [Project structure, lifecycle, and bundling](#)
- [Mini app standardization](#)
- [Mini apps and super apps](#)
- [What are H5 and QuickApp?](#)

index

layout: collection override:tags: [] date: 2021-03-03

updated: 2021-03-03

pathName: mini-apps

What are mini apps?

{% 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 %}

Building blocks and compatibility

Mini apps are small (commonly [2-4 MB](#)) apps that require a [super app](#) to run. What they have in common, independent of the super app, is that they are built with ("dialects" of) the web technologies HTML, CSS, and JavaScript. The runtime of a mini app is a [WebView](#) in the super app, not the underlying operating system, which makes mini apps cross platform. The same mini app can run in the same super app, no matter if the super app runs on Android, iOS, or another OS. However, not all mini apps can run in all super apps, more on this [later](#).

Discovery

Mini apps are often discovered *ad-hoc* via branded 2D barcodes, which solves an important offline-to-online challenge, for example, getting from a physical restaurant menu to a payment mini app, or from a physical e-scooter to a rental mini app. The image below shows an example of such a branded 2D barcode for [WeChat's demo mini app](#). When the code is scanned with the WeChat super app, the mini app launches directly. Other super apps will typically not be able to recognize the barcode.

```
{% Img src="image/tcFciHGuf3MxnTr1y5ue01OGLBn2/SOisfOqKQWr0GZZvUaqn.jpg",  
alt="WeChat-branded 2d barcode.", width="250", height="250" %}
```

Scanning this 2d barcode with the WeChat app launches a demo mini app.

Mini apps can also be discovered through regular in-app search in the super app, be shared in chat messages, or be part of a news item in a news feed. Some super apps have the notion of verified accounts that can contain mini apps in their profiles. Mini apps can be highlighted when they are physically geographically close, like the mini app of a business in front of which the user stands, or virtually close, like when the user gets directions on a map shown in the super app. Frequently used mini apps are available in an app drawer that in many super apps can be accessed through a swipe down gesture or through a special section in the super app menu.

The user experience

All super apps have more or less the same user interface for mini apps. A themeable top bar with the mini app's name, and, in the upper corner of the screen, a close button on the far right preceded by an action menu that provides access to common features like sharing the app, adding it to a favorites list or the home screen, reporting abusive apps, providing feedback, and settings. The screenshot below shows a shopping mini app running in the context of the Alipay super app with the action menu opened.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/PkjzF8AyxDVIAMZmhVrr.jpg",  
alt="The Alipay super app running a shopping mini app with highlighted top bar, action  
menu button, and close button. The action menu is opened.", width="300",  
height="649" %}
```

Opened action menu of a shopping mini app running in the Alipay super app.

UI paradigms

Usually there is a bottom tab bar for the mini app's main navigation. Most [super app providers offer components](#) that help developers quickly implement common UI paradigms, such as carousels, accordions, progress bars, spinners, switches, maps, and so on. This also helps make the user experience between different mini apps consistent, which is encouraged by [WeChat's Mini Program Design Guidelines](#). This is similar to what Apple incentivizes with its [Apple Human Interface Guidelines](#), and Google with its [Design for Android](#) recommendations.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/nFVCU3HqKERzl7Lops6Q.png",  
alt="The Douyin demo mini app showcasing the Douyin slider (carousel) component  
with toggles for auto-advance, dot indicators, etc.", width="300", height="617" %}  
Douyin's slider (carousel) component with various options.
```

Serving

Rather than being served piece by piece as separate resources, mini apps are served as encrypted packaged apps, that is, as archives that contain all resources in just one file. Unlike regular web apps, they are also not served from the particular origin of the mini app creator, but from the super app provider directly. They can still access APIs from the servers of the mini app creator, but the core resources (commonly referred to as the app shell), must be served from the super app provider. Mini apps have to declare the origins they request additional data from.

Caching, updates, and deep-linking

Mini apps are kept in the cache of the super app, so the next time the user launches a cached mini app, it loads almost instantly. If there is an update, a new app package is loaded. The version number can be part of the launch URI (see [Discovery](#)), so the super app knows early on if the locally cached version is still current. The launch URI also optionally contains the desired page of the mini app, so deep-linking into specific pages of mini apps is possible. Via a sitemap, mini apps can declare which of their pages should be indexable by the super app provider's mini app crawler.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/ZKLNvNnm3Rr6aBmYbons.png",  
alt="macOS Finder showing a folder containing cached WeChat mini app `.wxapkg`  
files.", width="800", height="465" %}
```

Mini apps are cached as encrypted packaged apps.

Security and permissions

Mini apps are reviewed by the super app provider, which means users perceive them as more secure than web apps. They need to declare their potentially required permissions beforehand in a manifest or mini app configuration file, which, for some providers, also requires explanations for why each permission is needed. Mini apps can of course still lie, but they would have a hard time justifying why they are, for example, [trying to access motion sensors](#) without a reason that is apparent to the user. The incentive to fingerprint the user is notably a lot lower compared to the web, since the user is typically already logged in to the super app anyway (see [Identity, payment, and social graph](#)).

Whenever a mini app performs an operation that requires a special permission, a prompt is shown to the user that, if enforced by the platform, also includes the usage justification, as stated by the developer. The screenshot below shows the [Douyin demo mini app](#) as it asks the user for permission to share their location. In some super apps, there is also an imperative API that mini apps can leverage to request permissions without immediately using them, or to only check the status of a permission. This may even include an API to open the central super app permission settings, which corresponds to [Chrome's Site Settings](#). Mini apps also have to declare beforehand the origins of all servers that they potentially will request data from.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/8To8DiUqnP4qqFfpPNqk.png",  
alt="The Douyin demo mini app showing a geolocation prompt with two options: 'Not  
Allowed' and 'Allowed'.", width="300", height="617" %}
```

The Douyin demo mini app asking for the geolocation permission.

Access to powerful features

The hosting super app offers access to powerful APIs via a JavaScript bridge that gets injected into the WebView offered by the super app (see [Building blocks and compatibility](#)). This JavaScript bridge provides hooks into the operating system's APIs. For example, a mini app JavaScript function like `getConnectedWifi()`—the capability of a mini app to obtain the name of the currently active Wi-Fi network—under the hood is facilitated through Android's `getConnectionInfo()` API or iOS' `CNCopyCurrentNetworkInfo()` API. Other examples of powerful device APIs exposed in common super apps are Bluetooth, NFC, iBeacon, GPS, system clipboard, orientation sensors, battery information, calendar access, phonebook access, screen brightness control, file system access, vibration hardware for physical feedback, camera and microphone access, screen recording and screenshot creation, network status, UDP sockets, barcode scanning, device memory information, and more.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/HNEKyoLVeq3lUKGXEZEZ.png",  
alt="The WeChat demo mini app showing a slider that controls the screen brightness of  
the device moved all the way to the maximum.", width="300", height="649" %}
```

The WeChat demo mini app setting the device's screen brightness to the maximum.

Access to cloud services

Many super apps also provide "serverless" access to cloud services of the super app provider that, apart from raw cloud computing and cloud storage, frequently also include higher-level tasks like text translation, object detection or content classification in images, speech recognition, or other Machine Learning tasks. Mini apps can be monetized with ads, which are commonly made available by super apps providers. Super app platforms usually also provide cloud analytics data, so mini app developers can better understand how users interact with their apps.

Identity, payment, social graph

A very important feature of mini apps is the identity and social graph information that is shared from the super app. Super apps like Douyin or WeChat started as social networking sites in the broadest sense, where users have a (sometimes even government-verified) identity, a friend or follower network, and frequently also payment data stored. For example, a shopping mini app can (or sometimes even must) process any payments directly through the payment APIs of the super app and, upon user consent, can obtain user data like their shipping address, phone number, and full name, all without ever having to force the user to painfully fill out forms. Below, you can see the Walmart mini app running in WeChat, opened for the very first time, greeting me with a familiar face.

{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/HzPx6ZSqQWvsUDT04ex2.png", alt="The Walmart mini app showing the author's face and name on the 'Me' tab.", width="300", height="649" %}

The Walmart mini app with a personalized "Me" view on the first visit.

Mini apps can get highly popular by letting people share their achievements like highscores in a game and challenge their contacts through status updates. The mini app is then only a tap away, so users can enter into competition without any friction and the mini app thereby grow its reach.

{% Aside 'success' %} In the next chapter, you will learn [what sets mini apps apart from H5 apps and QuickApp](#). {% endAside %}

Acknowledgements

This article was reviewed by [Joe Medley](#), [Kayce Basques](#), [Milica Mihajlija](#), [Alan Kent](#), and Keith Gu.

Other mini app runtime environments

{% 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 %}

More than just on mobile

In countries like China, mini apps have taken the market by storm. Apart from mobile devices, where mini apps are omnipresent and which are their natural habitat, mini apps have started to conquer other runtime environments like cars and the classic desktop.

Mini apps in cars

In July 2020 the German car maker BMW Group [announced](#) a collaboration with Tencent branded as WeScenario, which, [according to Tencent](#), will be rolled out to *"30 leading auto companies in the world, and [bring the WeScenario] ecosystem of social, content and services to more than 110 mainstream automobile models"*.

{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/AX07xQIEHL7MDjPo1U1H.jpg", alt="Dashboard of a Tencent car showing two rows of mini app icons.", width="800", height="533" %}

Landing page of Tencent WeScenario (Source: [BMW](#)).

Mini apps on the desktop

Mini apps in WeChat Desktop

Using the WeChat desktop client available for [macOS](#) and [Windows](#)), it is possible to run WeChat mini apps on the desktop. (Make sure to *not* load the macOS [version from the App Store](#) if you are doing research and want the full experience, since it is more limited.)

To test it on macOS, share a mini app from a mobile device with yourself via the "File Transfer" account. This will result in a message that you can then open on the desktop client. In most cases, the mini app will then be directly clickable and runnable. In other cases, you have to forward the chat history to yourself again from a mobile device.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/qLbYBpAoSvbc1Qjc9Ub0.png",  
alt="The WeChat macOS desktop client showing a chat with oneself with a shared mini  
app and a chat history as the two visible messages.", width="800", height="602" %}  
Sharing a mini app with oneself in the WeChat macOS desktop client.
```

On Windows, the workaround to share mini apps with oneself is not necessary, since there is a dedicated mini apps panel that shows the user's recently used mini apps and also includes an app search where new mini apps can be discovered.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/NQCoGaAWcbuiO37dCNY5.png",  
alt="The mini app panel in the WeChat Windows client showing the user's recently used  
mini apps.", width="800", height="531" %}  
The mini app panel in the WeChat Windows client.
```

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/H5nqmnnoK9JjLcu8mWDSH.png",  
alt="The mini app search in the WeChat Windows client showing mini apps listed in  
various categories like games, business, education, etc.", width="800", height="576"  
%}  
The mini app search in the WeChat Windows client.
```

WeChat mini apps on the desktop naturally integrate with the operating system. On both macOS and Windows, they get their own entry in the multitasking bar and have their own taskbar icon. While on macOS, there is an option to be kept in the Dock, the icon disappears the moment the WeChat client app gets closed. On Windows, mini app icons can be pinned to the taskbar, but cannot be launched. On macOS, the title of the app is always "WeChat" and not the actual title of the app, whereas the title is displayed correctly on Windows.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/nrvwtwOWot6eZI40evfp.png",  
alt="The macOS multitask switcher includes mini apps alongside regular macOS app.",  
width="800", height="79" %}  
The Starbucks app is a mini app and can be multitasked to like any regular macOS app.
```

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/fG4cpJgpyXeqvjn4IF3v.png",  
alt="The Starbucks mini app icon on the macOS Dock with a WeChat title.",  
width="646", height="412" %}  
Mini apps on macOS have WeChat as their title.
```

Most mini apps are not optimized for desktop yet and run in a fixed, non-resizable window that includes the well-known UI affordances and permission prompts as on mobile (see [The user experience](#)).

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/uorcciuZIL8sadxTjXbv.png",  
alt="The Starbucks mini app running on macOS asking for the user profile permission  
which the user can grant via a prompt shown at the bottom.", width="300",  
height="484" %}
```

The Starbucks mini app running on macOS asking for the user profile permission.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/UGufvRnoaAGm8A3qQG4a.png",  
alt="The Starbucks mini app running on macOS showing the home screen of the app.",  
width="300", height="484" %}
```

The Starbucks mini app running on macOS in a fixed, non-resizable window.

Responsive mini apps that are optimized for the desktop (apart from for mobile) can be displayed in a wider window that on macOS is currently still fixed, but that on Windows is flexibly resizable.

```
{% Img  
src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/cUMqQ75zPeCDNTcCWF5D.png",  
alt="The WeChat components demo app in a responsive app window that can be  
resized and that by default is wider than the usual mobile screen.", width="800",  
height="620" %}
```

The WeChat components demo app in a responsive app window.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/rPbojIjBGKbNtDviB0MJ.png",  
alt="The WeChat components demo app in a narrow window showing three boxes A, B,  
and C stacked on top of each other.", width="300", height="614" %}
```

The WeChat components demo app in a narrow app window.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/xnlacc8Xm1eNMCuajYYL.png",  
alt="The WeChat components demo app in a wide window showing three boxes A, B,  
and C with A stacked on top of B and C on the side.", width="600", height="565" %}
```

The WeChat components demo app in a wide app window.

Mini app permission settings on macOS can be changed via the context menu. On Windows, this does not seem to be possible and the location reported by the demo app appears to be the coarse location that Windows allows apps to obtain without asking for permission.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/rHyiTGbMauqbSFWBFIMW.png",  
alt="The WeChat components demo app running on macOS showing two checkboxes  
for the location and user info permission.", width="500", height="384" %}
```

WeChat mini app settings on macOS.

Mini apps in the 360 Secure Browser

360 Secure Browser (360 浏览器) is a web browser developed by the company Qihoo. Apart from [iOS and Android](#), the browser is also available for [Windows](#), [macOS](#), and [Linux](#). On Windows, it is capable of running special [360 mini apps](#). The [developer documentation](#) and the [API](#) are comparable to those of other vendors; however, 360 does not offer dedicated DevTools. Instead, developers need to create their mini apps in an IDE of their own choosing and can then test them in the browser using a special development mode. Debugging happens through Chrome Dev Tools. A [demo app](#) is available for getting started.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/IdHpN4GhVWDZ5gmSZ4Jb.png",  
alt="A 360 mini app running in 360 Secure Browser being debugged with Chrome Dev  
Tools.", width="800", height="402" %}  
Debugging a 360 mini app using Chrome Dev Tools.
```

360 mini apps can run in fullscreen mode and do appear as separate entries in the multitasking bar. Via the context menu, a home screen icon can be added that allows for mini apps to be launched from the desktop.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/M4MPV6TXwsLC6IFn9Jgi.png",  
alt="A 360 video mini app running in fullscreen mode showing various thumbnails of  
videos to watch.", width="800", height="402" %}  
360 mini app running in fullscreen mode.
```

Web-based mini apps

There are some mini app platforms that are web-based, but that depend on the presence of a special WebView to unlock their full potential.

LINE

[LINE](#) is an app for instant communications on electronic devices such as smartphones, tablet computers, and personal computers. In addition, LINE is a platform providing various services including digital wallet, news stream, video on demand, and digital comic distribution. The service is a subsidiary of Korean internet search engine company, [Naver Corporation](#).

Since LINE [mini apps](#) is [essentially just a regular web app](#) (see [sample app](#)) that pulls in the [LINE Front-end Framework](#) (LIFF), it can also be accessed outside of the main LINE app through special [permanent links](#) ([example](#)). However, not all APIs are available in such cases. Examples of not available in the browser APIs include the `liff.scanCode()` method for reading QR codes or Bluetooth-related APIs like `liff.bluetooth.getAvailability()`. To get a feel for the platform, you can test the [LINE Playground app](#) in the browser and the LINE app if you have a LINE account.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/YlIFjXgZhq8ROMrPA1BO.png",  
alt="The LINE Playground demo app running on an iOS device showing `liff.getOS()`  
returning 'ios'.", width="300", height="649" %}
```

The LINE Playground demo app running on an iOS device.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/UPwJbVhssGfA4IQ89kEo.png",  
alt="The LINE Playground demo app running in the web browser showing `liff.getOS()`  
returning 'web'.", width="800", height="510" %}
```

The LINE Playground demo app running in the web browser.

Google Spot

The [Google Spot Platform](#) allows developers to set up a Spot on [Google Pay](#)—a digital storefront that they can create, brand, and host however they choose. It is discoverable both online as well as through physical barcodes. Users can easily share a "Spot" (as the app calls it) on their favorite messaging app or find it on Google Pay. A Spot is built using HTML and JavaScript, so existing investments into mobile websites or PWAs can be easily transformed into a Spot by "adding a few lines of JavaScript" according to the announcement post.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/T4OLZX8PIFjFq3gVzUmo.png",  
alt="The Eat.fit mini app running in the Google Pay super app showing the sign-in  
bottom sheet.", width="300", height="637" %}
```

The Eat.fit mini app running in the Google Pay super app (Source: [Google](#)).

Snap Minis

[Snap Inc.](#) is an American camera and social media company most known for its chat app [Snapchat](#).

Snap has announced [Snap Minis](#), bite-size utilities made for friends. They are built with HTML5, so they are easy to develop. Plus, they work for all Snapchatters, on any kind of device, with no installation required.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/vM5d0wK5fCQSV0VyP7HA.png",  
alt="The Atom Tickets mini app running in Snapchat showing three snapchat users  
reserving their seats in a movie theater.", width="320", height="470" %}
```

The Atom Tickets mini app running in Snapchat (Source: [Snap](#)).

VK Mini Apps

The Russian social networking platform [VK](#) runs a [mini app program](#) that allows developers to [build](#) mini apps that tie in deeply with the social network. VK mini apps work on both VK's platform-specific mobile apps, as well as on the desktop website. Apart from several of the brand's other

platforms like [Mail.ru](#), VK mini apps are also integrated in the [Atom browser](#).

```
{% Img src="image/8WbTDNrhLsU0EI80frMBGE4eMCD3/tZJgYpdjurBVfhInqWRo.webp",  
alt="The Все аптеки mini app running in VK.", width="460", height="948" %}
```

The Все аптеки mini app running in VK (Source: [VK](#)).

```
{% Aside 'success' %} Read on to see some examples of the mini app open source projects. {%  
endAside %}
```

Acknowledgements

This article was reviewed by [Joe Medley](#), [Kayce Basques](#), [Milica Mihajlija](#), [Alan Kent](#), and Keith Gu.

Mini app components

{% 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 %}

Web components

[Web components](#) started with the promise of letting developers piece them together and build great apps on top of them. Examples of such atomic components are GitHub's [time-elements](#), Stefan Judis' [web-vitals-element](#), or, shameless plug, Google's [dark mode toggle](#). When it comes to complete design systems, however, I have observed that people prefer to rely on a coherent set of components from the same vendor. An incomplete list of examples includes SAP's [UI5 Web Components](#), the [Polymer Elements](#), [Vaadin's elements](#), Microsoft's [FAST](#), the [Material Web Components](#), arguably the [AMP components](#), and many more. Due to a number of factors out of scope for this article, a lot of developers, however, have also flocked to frameworks like [React](#), [Vue.js](#), [Ember.js](#), etc. Rather than giving the developer the freedom to choose from any of these options (or, dependent on your viewpoint, *forcing* them to make a technology choice), super app providers universally supply a set of components that developers must use.

Components in mini apps

You can think of these components like any of the component libraries mentioned above. To get an overview of the available components, you can browse [WeChat's component library](#), [ByteDance's components](#), [Alipay's components](#), [Baidu's](#), and [Quick App components](#).

Earlier [I showed](#) that while, for example, WeChat's `<image>` is a web component under the hood, not all of these components are technically web components. Some components, like `<map>` and `<video>`, are rendered as [OS-built-in components](#) that get layered over the WebView. For the developer, this implementation detail is not revealed, they are programmed like any other component.

As always, the details vary, but the overall programming concepts are the same across all super app providers. An important concept is data binding, as shown before in [Markup languages](#). Generally, components are grouped by function, so finding the right one for the job is easier. Below is an example from Alipay's categorization, which is similar to the component grouping of other vendors.

- View containers

- view
- swiper
- scroll-view
- cover-view
- cover-image
- movable-view
- movable-area

- Basic content

- text
- icon
- progress
- rich-text

- Form components

- button
- form
- label
- input
- textarea
- radio
- radio-group
- checkbox
- checkbox-group
- switch
- slider
- picker-view
- picker

- Navigation

- navigator

- Media components

- image
- video

- Canvas

- canvas

- Map

- map

- Open components

- web-view
- lifestyle

- `contact-button`
- Accessibility
 - `aria-component`

Below, you can see Alipay's `<image>` used in an `a:for` directive (see [List rendering](#)) that loops over an image data array provided in `index.js`.

```
/* index.js */
Page({
  data: {
    array: [
      {
        mode: "scaleToFill",
        text: "scaleToFill",
      },
      {
        mode: "aspectFit",
        text: "aspectFit",
      },
    ],
    src: "https://images.example.com/sample.png",
  },
  imageError(e) {
    console.log("image", e.detail.errMsg);
  },
  onTap(e) {
    console.log("image tap", e);
  },
  imageLoad(e) {
    console.log("image", e);
  },
});
```

```
<!-- index.axml -->
<view class="page">
  <view class="page-section" a:for="{% raw %}{{array}}{% endraw %}" a:for-item="item">
    <view class="page-section-demo" onTap="onTap">
      <image
        class="image"
        mode="{% raw %}{{item.mode}}{% endraw %}"
```

```

        onTap="onTap"
        onError="imageError"
        onLoad="imageLoad"
        src="{% raw %}{{src}}{% endraw %}"
        lazy-load="true"
        default-source="https://images.example.com/loading.png"
    />
</view>
</view>
</view>

```

Note the data binding of the `item.mode` to the `mode` attribute, the `src` to the `src` attribute, and the three event handlers `onTap`, `onError`, and `onLoad` to the functions of the same name. As shown [before](#), the `<image>` tag internally gets converted into a `<div>` with a placeholder of the image's final dimensions, optional lazy loading, a default source, etc.

The available configuration options of the component are all listed in the [documentation](#). An embedded-in-the-docs [component preview with simulator](#) makes the code immediately tangible.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/t2Y6WWsRhp6LjThxROUo.png",
alt="Alipay component documentation with embedded component preview, showing a
code editor with simulator that shows the component rendered on a simulated iPhone
6.", width="800", height="510" %}
```

Alipay component documentation with embedded component preview.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/xu2F51XL9Z0M3Z9n6n2O.png",
alt="Alipay component preview running in a separate browser tab showing a code
editor with simulator that shows the component rendered on a simulated iPhone 6.",
width="800", height="514" %}
```

Alipay component preview popped out into its own tab.

Each component also has a QR code that can be scanned with the Alipay app that opens the component example in a self-contained minimal example.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/gaWqLG5GqeqqfbhWz8D1.png",
alt="Alipay's `image` component previewed on a real device after scanning a QR code
in the documentation.", width="300", height="649" %}
```

Preview of the Alipay `<image>` component on a real device after following a [QR code link](#) from the docs.

Developers can jump from the documentation straight into Alipay DevTools IDE by leveraging a proprietary URI scheme `antdevtool-tiny://`. This allows the documentation to link directly into a to-be-imported mini app project, so developers can get started with the component immediately.

Custom components

Apart from using the vendor-provided components, developers can also create custom components. The concept exists for [WeChat](#), [ByteDance](#), [Alipay](#), and [Baidu](#), as well as [Quick App](#). For example, a Baidu custom component consists of four files that must reside in the same folder: `custom.swan`, `custom.css`, `custom.js`, and `custom.json`.

The file `custom.json` denotes the folder contents as a custom component.

```
{
  "component": true
}
```

The file `custom.swan` contains the markup and `custom.css` the CSS.

```
<view class="name" bindtap="tap">{% raw %}{{name}}{% endraw %} {% raw %}{{age}}{% endraw %}</view>
```

```
.name {
  color: red;
}
```

The file `custom.js` contains the logic. The component lifecycle functions are `attached()`, `detached()`, `created()`, and `ready()`. The component can additionally also react on [page lifecycle events](#), namely `show()` and `hide()`.

```
Component({
  properties: {
    name: {
      type: String,
      value: "swan",
    },
  },
  data: {
    age: 1,
  },
  methods: {
    tap: function () {},
  },
})
```

```
lifetimes: {
  attached: function () {},
  detached: function () {},
  created: function () {},
  ready: function () {},
},
pageLifetimes: {
  show: function () {},
  hide: function () {},
},
});
```

The custom component can then be imported in `index.json`, the key of the import determines the name (here: `"custom"`) that the custom component can then be used with in `index.swan`.

```
{
  "usingComponents": {
    "custom": "/components/custom/custom"
  }
}
```

```
<view>
  <custom name="swanapp"></custom>
</view>
```

{% Aside 'success' %} Continue reading to learn about the [project structure, lifecycle, and bundling](#) of mini apps. {% endAside %}

Acknowledgements

This article was reviewed by [Joe Medley](#), [Kayce Basques](#), [Milica Mihajlija](#), [Alan Kent](#), and Keith Gu.

Concluding thoughts about mini apps from a web developer

{% 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 %}

Where does this leave us?

Writing and researching mini apps has been quite a ride, but one that I do not regret. On the one hand, the success and the popularity of mini apps have proven their creators right about their approach. On the other hand, though, this success is geographically concentrated in regions where the few popular super apps are dominant, at least at the time of writing. What is undoubtedly true is that the ecosystem is highly fascinating and well worth a look. This collection of articles has provided deep-dives into many of the aspects that make a difference when using and creating mini apps. From the [DevTools](#) experience to the [mark-up](#), [styling](#), and [scripting](#) approaches, over to the [component model](#), and finally to the overall [architecture](#); mini apps provide learning and inspiration opportunities for app developers, and even so for those who purely aim for the web.

My initial experiments with [building a web application the mini app way](#) were successful. Future work will show to what extent this model is performant and flexible enough to cater for the many shapes that web apps can take. My current *ad-hoc* approach can be formalized by packaging up the relevant pieces of code in a dedicated library, `mini-app.js` if you will. What is interesting is that this kind of programming goes back all the way to `frameset`. Just that today it is about applications and not documents.

I see great potential for improvement with the entire web development experience by taking inspiration from the various mini apps DevTools. From the easy [\(remote\) on-device testing feature](#) to the packaging and [building](#) experience; the integration of the [IDE](#) with the DevTools environment offers a lot of starting points for making developers' lives easier.

Closing thoughts

From a features point of view, the web is becoming more and more powerful with each release of essentially *any* browser. The ever-growing [list of capabilities](#) makes use cases possible on the web that were unthinkable a mere year ago. At the same time, the need for [mini apps standardization](#) shows that developers are not willing or able to build the same mini app for each super app. On the horizon maybe there is a desire for an abstraction layer on the browser level that allows for mini apps to run on the web, while noting that the web is not immune from fragmentation, especially when looking at different browser vendors and what they choose to implement and what not. Concluding, I am looking forward to seeing where all this is headed. Thinking outside of the box and taking input and inspiration from outside of one's own bubble can definitely help when building a better future on the web.

{% Aside 'success' %} Congratulations, you have reached the end of the [mini apps collection](#). {% endAside %}

Acknowledgements

This article was reviewed by [Joe Medley](#), [Kayce Basques](#), [Milica Mihajlija](#), [Alan Kent](#), and Keith Gu.

Mini app DevTools

{% 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 %}

The developer experience

Now that I have covered mini apps *per se*, I want to focus on the developer experience for the various super app platforms. Mini app development on all platforms happens in IDEs that are provided for free by the super app platforms. While there are more, I want to focus on the four most popular ones, and a fifth for Quick App for comparison.

Mini app IDEs

Like the super apps, the majority of the IDEs are available only in Chinese. You actually want to make sure that you install the Chinese version and not a sometimes available English (or overseas) version, since it might not be up-to-date. If you are a macOS developer, be aware that not all IDEs are signed, which means macOS refuses to run the installer. You can, **at your own risk**, bypass this as [outlined by Apple help](#).

- [WeChat DevTools](#)
- [Alipay DevTools](#)
- [Baidu DevTools](#)
- [ByteDance DevTools](#)
- [Quick App DevTools](#)

Mini app starter projects

To get started quickly with mini app development, all super app providers offer demo apps that can be downloaded and tested immediately, and that are sometimes also integrated in the "New Project" wizards of the various IDEs.

- [WeChat demo](#)
- [Alipay demo](#)
- [Baidu demo](#)
- [ByteDance demo](#)
- [Quick App demo](#)

Development flow

After launching the IDE and loading or creating a (demo) mini app, the first step is always to log in. Usually you just need to scan a QR code with the super app (where you are already logged in) that is generated by the IDE. Very rarely do you have to enter a password. Once you are logged in, the IDE knows your identity and lets you start programming, debugging, testing, and submitting your app for review. In the following, you can see screenshots of the five IDEs mentioned in the paragraph above.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/YvjvIB82SfPqHBI56Rz4.png",  
alt="WeChat DevTools application window showing simulator, code editor, and  
debugger.", width="800", height="463" %}
```

WeChat DevTools with simulator, code editor, and debugger.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/iLbYZFZ9ec245segsIKk.png",  
alt="Alipay DevTools application window showing code editor, simulator, and  
debugger.", width="800", height="454" %}
```

Alipay DevTools with code editor, simulator, and debugger.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/svoq0p6GO1PCT0k0bdba.png",  
alt="Baidu DevTools application window showing simulator, code editor, and  
debugger.", width="800", height="540" %}
```

Baidu DevTools with simulator, code editor, and debugger.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/myapOcgYOwsEumeFbL3A.png",  
alt="ByteDance DevTools application window showing simulator, code editor, and  
debugger.", width="800", height="561" %}
```

ByteDance DevTools with simulator, code editor, and debugger.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/dUsfozY82E2YhtlxNJz9.png",  
alt="Quick App DevTools application window showing code editor, simulator, and  
debugger.", width="800", height="485" %}
```

Quick App DevTools with code editor, simulator, and debugger.

As you can see, the fundamental components of all IDEs are very similar. You always have a code editor based on the [Monaco Editor](#), the same project that also powers [VS Code](#). In all IDEs, there is a debugger based on the [Chrome DevTools frontend](#) with some modifications, more on those later

(see [Debugger](#)). The IDEs *per se* are implemented either as [NW.js](#) or as [Electron](#) apps, the simulators in the IDEs are realized as an [NW.js](#) `<webview>` tag or [Electron](#) `<webview>` tag, which in turn are based on a [Chromium](#) `<webview>` tag. If you are interested in the IDE internals, you can oftentimes simply inspect them with Chrome DevTools with the keyboard shortcut Control+Alt+I (or Command+Option+I on Mac).

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/yXcGmOhbi3xrcMIDhw1t.png",  
alt="Chrome DevTools used to inspect Baidu's DevTools showing the simulator's  
webview tag in the Chrome DevTools' Elements panel.", width="800", height="504" %}  
Inspecting Baidu DevTools with Chrome DevTools reveals that the simulator is realized  
as an Electron <webview> tag.
```

Simulator and real device testing and debugging

The simulator is comparable with what you might know from Chrome DevTools' [device mode](#). You can simulate different Android and iOS devices, change the scale and device orientation, but also simulate various network states, memory pressure, a barcode reading event, unexpected termination, and dark mode.

While the built-in simulator suffices to get a rough feeling for how the app behaves, on-device testing, like with regular web apps, is irreplaceable. Testing an in-development mini app is just a QR code scan away. For example, in ByteDance DevTools, scanning a QR code dynamically generated by the IDE with a real device leads to a cloud-hosted version of the mini app that can then immediately be tested on the device. The way this works for ByteDance is that the URL behind the QR code ([example](#)) redirects to a hosted page ([example](#)), that contains links with special URI schemes like, for example, `snssdk1128://`, to preview the mini app on the various ByteDance super apps like Douyin or Toutiao (here is an [example](#)). Other super app providers do not go through an intermediate page, but open the preview directly.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/p0CrYvh00oAV9AUzeoU6.png",  
alt="ByteDance DevTools showing a QR code that the user can scan with the Douyin  
app to see the current mini app on their device.", width="800", height="551" %}  
ByteDance DevTools showing a QR code that the user can scan with the Douyin app for  
immediate on-device testing.
```

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/LNOqaa8z2Oo4CZPa48Bw.png",  
alt="Intermediate landing page for previewing a ByteDance mini app in various of the  
company's super apps, opened on a regular desktop browser for reverse-engineering  
the process.", width="800", height="433" %}
```

Intermediate ByteDance landing page for previewing a mini app (opened on a desktop browser to show the flow).

An even more compelling feature is cloud-based preview remote debugging. After simply scanning a special likewise IDE-generated QR code, the mini app opens on the physical device, with a Chrome DevTools window running on the computer for remote debugging.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/N5Crv3ryZ3bCNFMv7Lir.png",  
alt="A mobile phone running a mini app with parts of the UI highlighted by the  
ByteDance DevTools debugger running on a laptop inspecting it.", width="800",  
height="600" %}
```

Wirelessly remote-debugging a mini app on a real device with ByteDance DevTools.

Debugger

Elements debugging

The mini app debugging experience is very familiar to anyone who has ever worked with Chrome DevTools. There are some important differences, though, that make the workflow tailored to mini apps. Instead of the Chrome DevTools' [Elements panel](#), mini app IDEs have a customized panel that is tailored to their particular dialect of HTML. For example, in the case of WeChat, the panel is called [Wxml](#), which stands for WeiXin Markup Language. In Baidu DevTools, it's called [Swan Element](#). ByteDance DevTools calls it [Bxml](#). Alipay names it [AXML](#), and Quick App references the panel simply as [UX](#). I will dive into these markup languages [later](#).

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/k1FO68wZhzpRNVQl2bN3.png",  
alt="Inspecting an image with WeChat DevTools' 'Wxml' panel. It shows that the tag in  
use is an `image` tag.", width="800", height="572" %}
```

Inspecting an `<image>` element with WeChat DevTools.

Custom elements under the hood

Inspecting the WebView on a real device via [about://inspect/#devices](#) reveals that WeChat DevTools was deliberately hiding the truth. Where WeChat DevTools showed an `<image>`, the actual thing I am looking at is a custom element called `<wx-image>` with a `<div>` as its only child. It is interesting to note that this custom element does not use [Shadow DOM](#). More on these components [later](#).

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/on9Ty46RyteTI6QTc2vk.png", alt="Inspecting a WeChat mini app running on a real device with Chrome DevTools. Where WeChat DevTools reported I am looking at an `image` tag, Chrome DevTools reveals I am actually dealing with a `wx-image` custom element.", width="800", height="385" %}
```

Inspecting an `<image>` element with WeChat DevTools reveals that it is actually a `<wx-image>` custom element.

CSS debugging

Another difference is the new length unit `rpx` for responsive pixel in the various dialects of CSS (more on this unit [later](#)). WeChat DevTools uses device-independent CSS length units to make developing for different device sizes more intuitive.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/bf4YBcscBzbmQtjig7lj.png", alt="Inspecting a view with a specified top and bottom padding of `200rpx` in WeChat DevTools.", width="800", height="486" %}
```

Inspecting the padding specified in responsive pixels (`200rpx 0`) of a view with WeChat DevTools.

Performance auditing

Performance is front and center for mini apps, so it is no surprise that WeChat DevTools and some other DevTools have an integrated Lighthouse-inspired auditing tool. The focus areas of the audits are Total, Performance, Experience, and Best Practice. The view of the IDE can be customized. In the screenshot below I have temporarily hidden the code editor to have more screen real estate for the audit tool.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/q8Rze6pR9mpDXw9VCaEs.png", alt="Running a performance audit with the built-in audit tool. The scores show Total, Performance, Experience, and Best Practice, each 100 out of 100 points.", width="800", height="485" %}
```

The built-in Audit tool in WeChat DevTools showing Total, Performance, Experience, and Best Practice.

API mocking

Rather than requiring the developer to set up a separate service, mocking API responses is directly part of WeChat DevTools. Via an easy-to-use interface the developer can set up API endpoints and the desired mock responses.

{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/olJmRvdl0zkoWZiiOkiP.png", alt="Setting up a mock response for an API endpoint in WeChat DevTools.", width="800", height="485" %}
WeChat DevTools' integrated API response mocking feature.

{% Aside 'success' %} Read on to learn about the [mark-up, styling, and scripting of mini apps](#). {% endAside %}

Acknowledgements

This article was reviewed by [Joe Medley](#), [Kayce Basques](#), [Milica Mihajlija](#), [Alan Kent](#), and Keith Gu.

Applying the mini app programming principles to an example project

{% 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 %}

The app domain

To show the [mini app way of programming](#) applied to a web app, I needed a small but complete enough app idea. [High-intensity interval training](#) (HIIT) is a cardiovascular exercise strategy of alternating sets of short periods of intense anaerobic exercise with less intense recovery periods. Many HIIT trainings use HIIT timers, for example, this [30 minute online session](#) from [The Body Coach TV](#) YouTube channel.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/tUI2jNm2bFqGBAFF5s63.png",  
alt="HIIT training online session with green high intensity timer.", width="800",  
height="450" %}  
Active period.
```

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/FMUgX3WJ4ZfQX38zHP75.png",  
alt="HIIT training online session with red low intensity timer.", width="800",  
height="450" %}  
Resting period.
```

HIIT Time example app

For this chapter, I have built a basic example of such a HIIT timer application aptly named "HIIT Time" that lets the user define and manage various timers, always consisting of a high and a low intensity interval, and then select one of them for a training session. It is a responsive app with a

navbar, a tabbar, and three pages:

- **Workout:** The active page during a workout. It lets the user select one of the timers and features three progress rings: the number of sets, the active period, and the resting period.
- **Timers:** Manages existing timers and lets the user create new ones.
- **Preferences:** Allows toggling sound effects and speech output and selecting language and theme.

The following screenshots give an impression of the application.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/9RYkQ17tIEy79NIAIFfP.svg",  
alt="HIIT Time example app in portrait mode.", width="800", height="450" %}  
HIIT Time "Workout" tab in portrait mode.
```

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/SNHMWFvHtCYHEfC9SHPI.svg",  
alt="HIIT Time example app in landscape mode.", width="800", height="450" %}  
HIIT Time "Workout" tab in landscape mode.
```

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/f7uqTk1PNMVaHob7FDzA.svg",  
alt="HIIT Time example app showing management of a timer.", width="800",  
height="450" %}  
HIIT Time timer management.
```

App structure

As outlined above, the app consists of a navbar, a tabbar, and three pages, arranged in a grid. Navbar and tabbar are realized as iframes with a `<div>` container in between them with three more iframes for the pages, out of which one is always visible and dependent on the active selection in the tabbar. A final iframe pointing to `about:blank` serves for dynamically created in-app pages, which are needed for modifying existing timers or creating new ones. I call this pattern multi-page single-page app (MPSPA).

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/Rv14TNs1kU0bpW5kv5bq.png",  
alt="Chrome DevTools view of the HTML structure of the app showing that it consists of  
six iframes: one for the navbar, one for the tabbar, and three grouped ones for each  
page of the app, with a final placeholder iframe for dynamic pages.", width="800",  
height="244" %}
```

The app consists of six iframes.

Components-based lit-html markup

The structure of each page is realized as [lit-html](#) scaffold that gets dynamically evaluated at runtime. For a background on lit-html, it is an efficient, expressive, extensible HTML templating library for JavaScript. By using it directly in the HTML files, the mental programming model is directly output-oriented. As a programmer, you write a template of what the final output will look like, and lit-html then fills the gaps dynamically based on your data and hooks up the event listeners. The app makes use of third-party custom elements like [Shoelace's](#) `<sl-progress-ring>` or a self-implemented custom element called `<human-duration>`. Since custom elements have a declarative API (for example, the `percentage` attribute of the progress ring), they work well together with lit-html, as you can see in the listing below.

```
<div>
  <button class="start" @click="${eventHandlers.start}" type="button">
    ${strings.START}
  </button>
  <button class="pause" @click="${eventHandlers.pause}" type="button">
    ${strings.PAUSE}
  </button>
  <button class="reset" @click="${eventHandlers.reset}" type="button">
    ${strings.RESET}
  </button>
</div>

<div class="progress-rings">
  <sl-progress-ring
    class="sets"
    percentage="${Math.floor(data.sets/data.activeTimer.sets*100)}"
  >
    <div class="progress-ring-caption">
      <span>${strings.SETS}</span>
      <span>${data.sets}</span>
    </div>
  </sl-progress-ring>
</div>
```

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/Toz6JmkCQVt7WLscSnIP.png",  
alt="Three buttons and a progress ring.", width="300", height="244" %}
```

Rendered section of the page corresponding to the mark-up above.

Programming model

Each page has a corresponding `Page` class that fills the lit-html markup with life by providing implementations of the event handlers and providing the data for each page. This class also supports lifecycle methods like `onShow()`, `onHide()`, `onLoad()`, and `onUnload()`. Pages have access to a data store that serves for sharing optionally persisted per-page state and global state. All strings are centrally managed, so internationalization is built in. Routing is handled by the browser essentially for free, since all the app does is toggle iframe visibility and for dynamically created pages change the `src` attribute of the placeholder iframe. The example below shows the code for closing a dynamically created page.

```
import Page from '../page.js';  
  
const page = new Page({  
  eventHandlers: {  
    back: (e) => {  
      e.preventDefault();  
      window.top.history.back();  
    },  
  },  
});
```

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/y82LVHSxUVAehgQIDbsb.png",  
alt="In-app page realized as an iframe.", width="500", height="272" %}
```

Navigation happens from iframe to iframe.

Styling

Styling of pages happens per-page in its own scoped CSS file. This means elements can usually just be directly addressed by their element names, since no clashes with other pages can occur. Global styles are added to each page, so central settings like the `font-family` or the `box-sizing` do not need to be declared repeatedly. This is also where the themes and dark mode options are defined. The listing below shows the rules for the Preferences page that lays out the various form elements on a grid.

```

main {
  max-width: 600px;
}

form {
  display: grid;
  grid-template-columns: auto 1fr;
  grid-gap: 0.5rem;
  margin-block-end: 1rem;
}

label {
  text-align: end;
  grid-column: 1 / 2;
}

input,
select {
  grid-column: 2 / 3;
}

```

{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/z3Op4O7OM5NQ1zz8Uah8.png", alt="HIIT Time app preferences page showing a form in grid layout.", width="500", height="312" %}}

Every page is its own world. Styling happens directly with the element names.

Screen wake lock

During a workout, the screen should not turn off. On browsers that support it, HIIT Time realizes this through a [screen wake lock](#). The snippet below shows how it is done.

```

if ('wakeLock' in navigator) {
  const requestWakeLock = async () => {
    try {
      page.shared.wakeLock = await navigator.wakeLock.request('screen');
      page.shared.wakeLock.addEventListener('release', () => {
        // Nothing.
      });
    }
  };
}

```

```
    } catch (err) {
      console.error(`${err.name}, ${err.message}`);
    }
  };
  // Request a screen wake lock...
  await requestWakeLock();
  // ...and re-request it when the page becomes visible.
  document.addEventListener('visibilitychange', async () => {
    if (
      page.shared.wakeLock !== null &&
      document.visibilityState === 'visible'
    ) {
      await requestWakeLock();
    }
  });
}
```

Testing the application

The HIIT Time application is available on [GitHub](#). You can play with the [demo](#) in a new window, or right in the iframe embed below, which simulates a mobile device.

<https://tomayac.github.io/hiit-time/#workout>

{% Aside 'success' %} The final chapter ends this collection on mini apps with a [conclusion](#). {% endAside %}

Acknowledgements

This article was reviewed by [Joe Medley](#), [Kayce Basques](#), [Milica Mihajlija](#), [Alan Kent](#), and Keith Gu.

Mini app markup, styling, and scripting

{% 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 %}

Markup languages

As outlined before, rather than with plain HTML, mini apps are written with dialects of HTML. If you have ever dealt with [Vue.js](#) text interpolation and directives, you will feel immediately at home, but similar concepts existed way before that in XML Transformations ([XSLT](#)). Below, you can see code samples from WeChat's [WXML](#), but the concept is the same for all mini apps platforms, namely Alipay's [AXML](#), Baidu's [Swan Element](#), ByteDance's [TTML](#) (despite the DevTools calling it Bxml), and Quick App's [HTML](#). Just like with Vue.js, the underlying mini app programming concept is the [model-view-viewmodel](#) (MVVM).

Data binding

Data binding corresponds to Vue.js' [text interpolation](#).

```
<!-- wxml -->
<view>{% raw %}{{message}}{% endraw %}</view>
```

```
// page.js
Page({
  data: {
    message: "Hello World!",
  },
});
```

List rendering

List rendering works like Vue.js [v-for directive](#).

```
<!-- wxml -->
<view wx:for="{% raw %}{{array}}{% endraw %}">{% raw %}{{item}}{% endraw %}</view>
```

```
// page.js
Page({
  data: {
    array: [1, 2, 3, 4, 5],
  },
});
```

Conditional rendering

Conditional rendering works like Vue.js' [v-if directive](#).

```
<!-- wxml -->
<view wx:if="{% raw %}{{view == 'one'}}{% endraw %}">One</view>
<view wx:elif="{% raw %}{{view == 'two'}}{% endraw %}">Two</view>
<view wx:else="{% raw %}{{view == 'three'}}{% endraw %}">Three</view>
```

```
// page.js
Page({
  data: {
    view: "three",
  },
});
```

Templates

Rather than requiring the imperative [cloning of the content](#) of an [HTML template](#), WXML templates can be used declaratively via the `is` attribute that links to a template definition.

```
<!-- wxml -->
<template name="person">
```

```
<view>
  First Name: {% raw %}{{firstName}}{% endraw %}, Last Name: {% raw %}{{lastName}}{% endraw
%}
</view>
</template>
```

```
<template is="person" data="{% raw %}{{...personA}}{% endraw %}"></template>
<template is="person" data="{% raw %}{{...personB}}{% endraw %}"></template>
<template is="person" data="{% raw %}{{...personC}}{% endraw %}"></template>
```

```
// page.js
Page({
  data: {
    personA: { firstName: "Alice", lastName: "Foo" },
    personB: { firstName: "Bob", lastName: "Bar" },
    personC: { firstName: "Charly", lastName: "Baz" },
  },
});
```

Styling

Styling happens with dialects of CSS. WeChat's is named [WXSS](#). For Alipay, theirs is called [ACSS](#), Baidu's simply [CSS](#), and for ByteDance, their dialect is referred to as [TTSS](#). What they have in common is that they extend CSS with responsive pixels. When writing regular CSS, developers need to convert all pixel units to adapt to different mobile device screens with different widths and pixel ratios. TTSS supports the `rpx` unit as its underlying layer, which means the mini app takes over the job from the developer and converts the units on their behalf, based on a specified screen width of `750rpx`. For example, on a Pixel 3a phone with a screen width of `393px` (and a device pixel ratio of `2.75`), responsive `200rpx` become `104px` on the real device when inspected with Chrome DevTools ($393\text{px} / 750\text{rpx} * 200\text{rpx} \approx 104\text{px}$). In Android, the same concept is called [density-independent pixel](#).

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/n26ptkMoSfiTDanfFh5F.png",
alt="Inspecting a view with Chrome DevTools whose responsive pixel padding was
specified with `200rpx` shows that it is actually `104px` on a Pixel 3a device.",
width="800", height="385" %}
```

Inspecting the actual padding on a Pixel 3a device with Chrome DevTools.

```
/* app.wxss */
.container {
  height: 100%;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: space-between;
  padding: 200rpx 0; /* ← responsive pixels */
  box-sizing: border-box;
}
```

Since components (see [later](#)) do not use shadow DOM, styles declared on a page reach into all components. The common stylesheet file organization is to have one root stylesheet for global styles, and individual per-page stylesheets specific to each page of the mini app. Styles can be imported with an `@import` rule that behaves like the `@import` CSS at-rule. Like in HTML, styles can also be declared inline, including dynamic text interpolation (see [before](#)).

```
<view style="color:{% raw %}{{color}}{% endraw %};" />
```

Scripting

Mini apps support a "safe subset" of JavaScript that includes support for modules using varying syntaxes that remind of [CommonJS](#) or [RequireJS](#). JavaScript code cannot be executed via `eval()` and no functions can be created with `new Function()`. The scripting execution context is [V8](#) or [JavaScriptCore](#) on devices, and V8 or [NW.js](#) in the simulator. Coding with ES6 or newer syntax is usually possible, since the particular DevTools automatically transpile the code to ES5 if the build target is an operating system with an older WebView implementation (see [later](#)). The documentation of the super app providers explicitly mentions that their scripting languages are not to be confused with and are distinct from JavaScript. This statement, however, refers mostly just to the way modules work, that is, that they do not support standard [ES Modules](#) yet.

As mentioned [before](#), the mini app programming concept is the [model-view-viewmodel](#) (MVVM). The logic layer and the view layer run on different threads, which means the user interface does not get blocked by long-running operations. In web terms, you can think of scripts running in a [Web Worker](#).

WeChat's scripting language is called [WXS](#), Alipay's [SJS](#), ByteDance's likewise [SJS](#). Baidu speaks of [JS](#) when referencing theirs. These scripts need to be included using a special kind of tag, for

example, `<wxs>` in WeChat. In contrast, Quick App uses regular `<script>` tags and the ES6 [JS](#) syntax.

```
<wxs module="m1">
  var msg = "hello world";
  module.exports.message = msg;
</wxs>

<view>{% raw %}{{m1.message}}{% endraw %}</view>
```

Modules can also be loaded via a `src` attribute, or imported via `require()`.

```
// /pages/tools.wxs
var foo = "'hello world' from tools.wxs";
var bar = function (d) {
  return d;
};
module.exports = {
  F00: foo,
  bar: bar,
};
module.exports.msg = "some msg";
```

```
<!-- page/index/index.wxml -->
<wxs src="../../tools.wxs" module="tools" />
<view>{% raw %}{{tools.msg}}{% endraw %}</view>
<view>{% raw %}{{tools.bar(tools.F00)}}{% endraw %}</view>
```

```
// /pages/logic.wxs
var tools = require("../tools.wxs");

console.log(tools.F00);
console.log(tools.bar("logic.wxs"));
console.log(tools.msg);
```

JavaScript bridge API

The JavaScript bridge that connects mini apps with the operating system makes it possible to use OS capabilities (see [Access to powerful features](#)). It also offers a number of convenience methods.

For an overview, you can check out the different APIs of [WeChat](#), [Alipay](#), [Baidu](#), [ByteDance](#), and [Quick App](#).

Feature detection is straightforward, since all platforms provide a (literally called like this)

`canIUse()` method whose name seems inspired by the website [caniuse.com](#). For example, ByteDance's `tt.canIUse()` allows for support checks for APIs, methods, parameters, options, components, and attributes.

```
// Testing if the `<swiper>` component is supported.
tt.canIUse("swiper");
// Testing if a particular field is supported.
tt.canIUse("request.success.data");
```

{% Aside 'success' %} The next chapter introduces [mini app components](#). {% endAside %}

Acknowledgements

This article was reviewed by [Joe Medley](#), [Kayce Basques](#), [Milica Mihajlija](#), [Alan Kent](#), and Keith Gu.

Mini app open source projects

{% 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 %}

kbone

The [kbone](#) project ([open source on GitHub](#)) implements an adapter that simulates a browser environment in the adaptation layer, so that code written for the web can run without changes in a mini app. Several starter templates (among them [Vue](#), [React](#), and [Preact](#)) exist that make the onboarding experience for web developers coming from these frameworks easier.

A new project can be created with the `kbone-cli` tool. A wizard asks what framework to initiate the project with. The code snippet below shows the Preact demo. In the code snippet below, the `mp` command builds the mini app, the `web` command builds the web app, and `build` creates the production web app.

```
npx kbone-cli init my-app
cd my-app
npm run mp
npm run web
npm run build
```

The code snippet below shows a simple counter component that then gets isomorphically rendered in a mini app and a web app. The overhead of the mini app is significant, purely judging from the DOM structure.

```
import { h, Component } from "preact";
import "./index.css";

class Counter extends Component {
  state = { count: 1 };
}
```

```

sub = () => {
  this.setState((prevState) => {
    return { count: --prevState.count };
  });
};

add = () => {
  this.setState((prevState) => {
    return { count: ++prevState.count };
  });
};

clickHandle = () => {
  if ("undefined" !== typeof wx && wx.getSystemInfoSync) {
    wx.navigateTo({
      url: "../log/index?id=1",
    });
  } else {
    location.href = "log.html";
  }
};

render({}, { count }) {
  return (
    <div>
      <button onClick={this.sub}>-</button>
      <span>{count}</span>
      <button onClick={this.add}>+</button>
      <div onClick={this.clickHandle}><div>
    </div>
  );
}
}

export default Counter;

```

{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/DDreycxzclMP8liUplyO.png", alt="The Preact kbone template demo app opened in WeChat DevTools. Inspecting the DOM structure shows a significant overhead compared to the web app.", width="800", height="664" %}

The Preact kbone template demo app opened in WeChat DevTools. Note the complex DOM structure and how the plus and minus buttons are actually not `<button>` elements.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/rerYiCnwc92DP7pM7WPc.png", alt="The Preact kbone template demo app opened in the web browser. Inspecting the DOM structure shows the to-be-expected markup based on the Preact component code.", width="800", height="496" %}
```

The Preact kbone template demo app opened in the web browser. Note the DOM structure.

kbone-ui

The [kbone-ui](#) project ([open source on GitHub](#)) is a UI framework that facilitates both mini app development as well as Vue.js development with kbone. The kbone-ui components emulate the look and feel of [WeChat's built-in mini app components](#) (also see [Components](#) above). A [demo](#) that runs directly in the browser lets you explore the available components.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/wEA4Hr2JyVfKgdHnpb5o.png", alt="Demo of the kbone-ui framework showing form-related components like radio buttons, switches, inputs, and labels.", width="320", height="438" %}
```

The kbone-ui demo showing form-related components.

WeUI

[WeUI](#) is a set of basic style libraries consistent with WeChat's default visual experience. The official WeChat design team has tailored designs for WeChat internal web pages and WeChat mini apps to make users' perception of use more uniform. It includes components such as `button`, `cell`, `dialog`, `progress`, `toast`, `article`, `actionsheet`, and `icon`. There are different versions of WeUI available like [weui-wxss](#) for WeChat mini apps styled with WXSS (see [Styling](#) above), [weui.js](#) for web apps, and [react-weui](#) for WeChat React components.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/V0xswCD3MjItrxALmy8n.png", alt="Demo of the WeUI framework showing form-related components, namely switches.", width="450", height="395" %}
```

The WeUI demo app showing switches.

Omi

[Omi](#) is a self-proclaimed frontend cross-frameworks framework ([open source on GitHub](#)). It merges Web Components, JSX, Virtual DOM, functional style, observer or Proxy into one framework with tiny size and high performance. Its aim is to let developers write components once and use them everywhere, such as Omi, React, Preact, Vue.js, or Angular. Writing Omi components is very intuitive and free of almost all boilerplate.

```
import { render, WeElement, define } from "omi";

define("my-counter", class extends WeElement {
  data = {
    count: 1,
  };

  static css = `
    span{
      color: red;
    }`;

  sub = () => {
    this.data.count--;
    this.update();
  };

  add = () => {
    this.data.count++;
    this.update();
  };

  render() {
    return (
      <div>
        <button onClick={this.sub}>-</button>
        <span>{this.data.count}</span>
        <button onClick={this.add}>+</button>
      </div>
    );
  }
});
```

```
render(<my-counter />, "body");
```

Omiu

[Omiu](#) is a cross framework UI component library ([open source on GitHub](#)) developed based on Omi, which outputs custom elements of standard web components.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/eoNhik827CE4TfaT9ZaV.png",  
alt="Demo of the Omiu framework showing form-related components, namely  
switches.", width="800", height="939" %}  
The Omiu demo app showing switches.
```

WePY

[WePY](#) is a framework that allows mini apps to support componentized development. Through pre-compilation, developers can choose their favorite development style to develop mini apps. The detailed optimization of the framework and the introduction of Promises and async functions all make the development of mini app projects easier and more efficient. At the same time, WePY is also a growing framework, which has largely absorbed some optimized frontend tools and framework design concepts and ideas, mostly from Vue.js.

```
<style lang="less">  
  @color: #4d926f;  
  .num {  
    color: @color;  
  }  
</style>  
  
<template>  
  <div class="container">  
    <div class="num" @tap="num++">{% raw %}{{num}}{% endraw %}</div>  
    <custom-component></custom-component>  
    <vendor-component></vendor-component>  
    <div>{% raw %}{{text}}{% endraw %}</div>  
    <input v-model="text" />  
  </div>  
</template>
```

```

<config>
  { usingComponents: { customComponent: '@components/customComponent', vendorComponent:
    'module:vendorComponent' } }
</config>

<script>
  import wepy from "@wepy/core";

  wepy.page({
    data: {
      num: 0,
      text: "Hello World",
    },
  });
</script>

```

{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/ooxZuXzVY9aHXmha36vM.png", alt="WePY 'getting started' documentation page showing the first steps to get going.", width="800", height="505" %}

WePY "getting started" documentation.

vConsole

The [vConsole](#) project provides a lightweight, extendable frontend developer tool for mobile web pages. It offers a DevTools-like debugger that can be injected directly into web apps and mini apps. A [demo](#) showcases the opportunities. The vConsole includes tabs for logs, system, network, elements, and storage.

{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/ow6Ffaol7fMCJDn1819t.png", alt="vConsole demo app. The vConsole opens at the bottom and has tabs for logs, system, network, elements, and storage.", width="800", height="505" %}

vConsole demo app showing the elements explorer.

weweb

The [weweb](#) project ([open source on GitHub](#)) is the underlying frontend framework of the [Hera](#) mini app framework that claims to be compatible with the syntax of WeChat mini apps, so you can write web applications in the way of mini apps. The documentation promises that if you already have a

mini app, you can run it in the browser thanks to weweb. In my experiments, this did not work reliably for current mini apps, most probably because the project has not seen updates recently leading its compiler to miss changes in the WeChat platform.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/jU9Od9IDqFlmuYzAtirn.png",  
alt="Documentation of the Hera mini app framework listing the WeChat APIs that it  
supports like `wx.request`, `wx.uploadFile`, etc.", width="800", height="505" %}  
Hera mini app framework documentation showing the list of supported WeChat APIs.
```

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/C7ZpPrFE4geW0dyIRpZ2.png",  
alt="The weweb demo mini app compiled with weweb to run in the browser showing  
form elements.", width="300", height="429" %}  
The weweb demo mini app compiled with weweb to run in the browser.
```

{% Aside 'success' %} In the next chapter, you will learn how to [program the mini app way](#). {%
endAside %}

Acknowledgements

This article was reviewed by [Joe Medley](#), [Kayce Basques](#), [Milica Mihajlija](#), [Alan Kent](#), and Keith Gu.

Programming the mini app way

{% 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.

Project structure, lifecycle, and bundling

{% 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 %}

Mini app project structure

As before with the markup languages, the styling languages, and the components; with the mini app project structure, too, the details like the file extensions or the default names vary. The overall idea, though, is the same for all super app providers. The project structure always consists of:

- A root file `app.js` that initializes the mini app.
- A configuration file `app.json` that *roughly* corresponds to a [web app manifest](#).
- An optional common style sheet file `app.css` with shared default styles.
- A `project.config.json` file that contains build information.

All the pages are stored in individual subfolders in a `pages` folder. Each page subfolder contains a CSS file, a JS file, an HTML file, and an optional configuration JSON file. All files must be named like their containing folder, apart from the file extensions. Like that, the mini app just needs a pointer to the directory in the `app.json` file (the manifest-like file), and can find all subresources dynamically. From the perspective of a web developer, mini apps are thus multi page apps.

```
├─ app.js           # Initialization logic
├─ app.json         # Common configuration
├─ app.css          # Common style sheet
├─ project.config.json # Project configuration
└─ pages            # List of pages
    ├─ index        # Home page
    │   ├─ index.css # Page style sheet
    │   ├─ index.js  # Page logic
    │   ├─ index.json # Page configuration
    │   └─ index.html # Page markup
```

└─ other	# Other page
└─ other.css	# Page style sheet
└─ other.js	# Page logic
└─ other.json	# Page configuration
└─ other.html	# Page markup

Mini app lifecycle

A mini app must be registered with the super app by calling the globally defined `App()` method. Referring to the project structure outlined [before](#), this happens in `app.js`. The mini app lifecycle essentially consists of four events: `launch`, `show`, `hide`, and `error`. Handlers for these events can be passed to the `App()` method in the form of a configuration object, which can also contain a `globalData` property that holds data that should be globally available across all pages.

```
/* app.js */
App({
  onLaunch(options) {
    // Do something when the app is launched initially.
  },
  onShow(options) {
    // Do something when the app is shown.
  },
  onHide() {
    // Do something when the app is hidden.
  },
  onError(msg) {
    console.log(msg);
  },
  globalData: "I am global data",
});
```

As usual, the individual details vary, but the concept is the same for [WeChat](#), [Alipay](#), [Baidu](#), [ByteDance](#), and also [Quick App](#).

Page lifecycle

Similar to the app lifecycle, the page lifecycle, too, has lifecycle events that the developer can listen for and react upon. These core events are `load`, `show`, `ready`, `hide`, and `unload`. Some platforms offer additional events like `pullDownRefresh`. Setting up the event handlers happens in the `Page()` method that is defined for each page. For the `index` or the `other` pages from the project structure [before](#), this would happen in `index.js` or `other.js` respectively.

```
/* index.js */
Page({
  data: {
    text: "This is page data.",
  },
  onLoad: function (options) {
    // Do something when the page is initially loaded.
  },
  onShow: function () {
    // Do something when the page is shown.
  },
  onReady: function () {
    // Do something when the page is ready.
  },
  onHide: function () {
    // Do something when the page is hidden.
  },
  onUnload: function () {
    // Do something when the page is closed.
  },
  onPullDownRefresh: function () {
    // Do something when the user pulls down to refresh.
  },
  onReachBottom: function () {
    // Do something when the user scrolls to the bottom.
  },
  onShareAppMessage: function () {
    // Do something when the user shares the page.
  },
  onPageScroll: function () {
    // Do something when the user scrolls the page.
  },
  onResize: function () {
```

```
// Do something when the user resizes the page.
},
onTabItemTap(item) {
  // Do something when the user taps the page's tab.
},
customData: {
  foo: "bar",
},
});
```

The build process

The build process of mini apps is abstracted away from the developer. Under the hood, it is using industry tools like the [Babel](#) compiler for transpilation and minification and the [postcss](#) CSS transformer. The build experience is comparable to that of [Next.js](#) or [create-react-app](#), where developers, if they explicitly choose not to, never touch the build parameters. The resulting processed files are finally signed, encrypted, and packaged in one or several (sub)packages that then get uploaded to the servers of the super app providers. Subpackages are meant for lazy loading, so a mini app does not have to be downloaded all at once. The packaging details are meant to be private and are not documented, but some package formats like WeChat's `wxapkg` format have been [reverse-engineered](#).

{% Aside 'success' %} The next chapter provides insights on the [mini app standardization effort](#).
{% endAside %}

Acknowledgements

This article was reviewed by [Joe Medley](#), [Kayce Basques](#), [Milica Mihajlija](#), [Alan Kent](#), and Keith Gu.

Mini app standardization

{% 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 %}

Mini app popularity

Mini apps have seen tremendous growth. WeChat mini apps as of June 2020 have reached [830 million active users](#), Alipay mini apps [401 million active users](#) as of April 2019, and Baidu mini apps in the same month [115 million active users](#). Effectively companies have traded building apps for the two operating systems iOS and Android and additionally the web for building apps for three or more [super apps platforms](#). The differences between each super app platform may not be as big as the differences between Android, iOS, and the web, but nevertheless they exist. Where on Android, iOS, and the web, we have seen cross-platform approaches like [Flutter](#), [Ionic](#), and [React Native \(for Web\)](#) gain popularity, in the mini apps ecosystem, we can see an effort led by the [MiniApps Ecosystem Community Group](#) with [members](#) from, among others, Alibaba, Baidu, ByteDance, Huawei, Intel, Xiaomi, China Mobile, Facebook, and Google to standardize aspects of mini apps.

Publications

Notable publications of the group so far include a [whitepaper](#), a [Comparison of APIs in MiniApps, W3C specs, and PWAs](#), and specifications and explainers on the following aspects:

- URI Scheme: [spec](#), [explainer](#)
- Lifecycle: [spec](#), [explainer](#)
- Manifest: [spec](#), [explainer](#)
- Packaging: [spec](#), [explainer](#)
- An exploration of [widget requirements](#)

W3C member and group participant Fuqiao Xue (W3C) has further published a [Comparison of MiniApps and web apps](#) on his own behalf, that is, not as an official group publication, but

nonetheless worth the read.

```
{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/NW2O3YZ3kxJPPeFr4Jw6.png",  
alt="The header of the MiniApp Standardization White Paper in a browser window.",  
width="800", height="540" %}
```

The MiniApp Standardization White Paper.

Formal launch of the W3C MiniApps Working Group

On January 19, 2021, the [MiniApps Working Group](#) was [formally launched](#) in the W3C. The group uses the spelling and capitalization "MiniApps" to distinguish the standardization effort from the technology. You can read the group's [charter](#) to get a feel for the planned work. Leaders of the group introduced the effort as follows:

“Currently, there are many variants of MiniApps developed by different vendors with different APIs. To enhance the interoperability between MiniApp platforms, mainstream MiniApp vendors including Alibaba, Baidu, Huawei, and Xiaomi have been working together in the [W3C Chinese Web Interest Group](#) since May 2019 and published a [MiniApp Standardization White Paper](#) in September 2019 as the initial standardization exploration for MiniApp technologies. As more global companies get interested in joining the MiniApp related discussion, the MiniApps Ecosystem Community Group launched during TPAC 2019 so that the global Web community can join the discussion.

“Based on extensive standardization requirements, W3C today announced the formal establishment of the MiniApps Working Group, dedicated to in-depth exploration and coordination of the diverse MiniApp ecosystem with W3C members and the public, and enhancing the interoperability of different MiniApp platforms, thereby maximizing the integration of MiniApps and the Web, reducing technical fragmentation and the learning cost of developers.

"Maximizing the integration of MiniApps and the Web" in particular sounds very interesting. As a curious member of the group, I look forward to seeing where this endeavor is headed.

{% Aside 'success' %} The following chapter looks at [alternative mini app runtime environments](#) apart from mobile devices. {% endAside %}

Acknowledgements

This article was reviewed by [Joe Medley](#), [Kayce Basques](#), [Milica Mihajlija](#), [Alan Kent](#), and Keith Gu.

Mini apps and super apps

When you look at applications on your phone, you probably have specific apps for specific tasks. You might have a banking app. You might have an app for buying public transit tickets. Likely you have an app for getting directions, and many more specialized apps. This post introduces you to the concept of a different kind of apps—mini apps—sometimes also called mini programs or applets. You will first learn about the background of various mini app platforms and their developer experience, and then focus on things the web can learn from mini apps. But before learning about mini apps, you first need to learn about super apps.

What are super apps?

Super apps serve as hosts to other apps that run within them: the so-called mini apps. Popular super apps are [WeChat](#) (微信) by Tencent, [Alipay](#) (支付宝) by Ant Group (an affiliate company of the Chinese Alibaba Group), the app of the search engine [Baidu](#) (百度), as well as ByteDance's [Douyin](#) (抖音), which you might know as TikTok (抖音). The first three are commonly also referred to as BAT, derived from **B**(aidu)**A**(libaba)**T**(encent). Super apps have taken the Chinese market by storm, which is why a lot of the examples in this article are Chinese.

```
{% Img
src="image/8WbTDNrhLsU0EI80frMBGE4eMCD3/UKmUgG231MtQ2nEo1P0K.PNG",
alt="List of recently launched mini apps in the WeChat super app.", width=300,
height=649 %}
```

The WeChat super app showing recently launched mini apps.

A few words about super app platforms

WeChat aims to make itself a one-stop shop to meet almost any need users might have in their daily lives. Alipay builds its platforms on top of its payment system, focusing on retail and financial services, including credit, loan, insurance, installment, and local life services. Baidu strives to transform its search engine from solely connecting people, services, and information into information-as-a-service through mini programs for travel, retail, ads, payment, and more. Last but not least Douyin wants to boost itself as a hub for social e-commerce and transform to more of an entertainment and shopping platform.

Installing super apps

Super apps are available on multiple operating systems. Note that the versions available in the official app stores may not always contain all features or be available in all locales. The links below point to links that work universally, but that may require loading from untrusted sources, so download and install the apps **at your own risk**. You typically need to create an account, which involves revealing your phone number. You might want to consider getting a burner phone. Be advised that many super apps only allow you to create a so-called overseas account, which does not have all features of a domestic account.

- **WeChat:** [iOS](#), [Android](#), [macOS](#), [Windows](#)
- **Baidu:** [iOS](#), [Android](#)
- **Alipay:** [iOS](#), [Android](#)
- **Douyin:** [iOS](#) (CN-only), [Android](#)

{% Aside %} Since the user interface of many super apps is Chinese-only, use the [Google Translate app](#) in camera mode with a secondary phone (given you have one) to understand what is going on if you do not speak Chinese. {% endAside %}

{% Img src="image/tcFciHGuF3MxnTr1y5ue01OGLBn2/kSLjHjkFgscBC2j2d6j9.png", alt="A secondary phone running Google Translate in camera mode live-translating the user interface of a Chinese mini app running on the primary phone.", width="300", height="520" %}

Using Google Translate in camera mode to live-translate a Chinese mini app.

{% Aside 'success' %} Read on to [learn more about mini apps](#) in the next chapter. {% endAside %}

Acknowledgements

This article was reviewed by [Joe Medley](#), [Kayce Basques](#), [Milica Mihajlija](#), [Alan Kent](#), and Keith Gu.

What are H5 and QuickApp?

{% 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 mini apps are not

Before I go into more detail on the developer experience of mini apps, I want to briefly mention and set apart two technologies that come up in the context of mini apps, H5 and Quick App.

H5

H5 apps (or pages) are commonly seen as the predecessor of mini apps. What people mean by H5 is essentially a well-designed mobile web app (or page) that can be shared easily on chat applications. H5 is a reference to the HTML5 umbrella of technologies that includes responsive design, snappy CSS animations, multimedia content, etc. The Chinese Wikipedia actually [redirects](#) from H5 to HTML5. A good example of a representative H5 page experience is the [WeChat H5 boilerplate](#) project's demo.

Quick App

[Quick App](#) is an industry alliance consisting of the following members:

- [vivo open platform](#)
- [Huawei Developer Alliance](#)
- [OPPO open platform](#)
- [Xiaomi Open Platform](#)
- [Lenovo Open Platform](#)
- [Gionee Open Platform](#)
- [Meizu Open Platform](#)

- [ZTE Developer Platform](#)
- [Nubian Open Platform](#)
- [OnePlus Open Platform](#)
- [Hisense Open Platform](#)
- [China Mobile Terminal Corporation](#)

While the technology of Quick App is comparable to "regular" mini apps (see [Building blocks and compatibility](#)), the discovery of Quick App is different. They are meant to be listed in stores, which come pre-installed on devices of the manufacturers in the alliance, but can also be shared by means of a deep link (see the [Quick App showcase](#)). They do not run in the context of a super app, but launch as seemingly self-contained full screen applications that feel deeply integrated into device. What happens in the background is that they are opened in a full screen view rendered by the operating system that provides the JavaScript bridge.

{% Aside 'success' %} The next chapter covers the [developer experience of mini apps](#). {% endAside %}

Acknowledgements

This article was reviewed by [Joe Medley](#), [Kayce Basques](#), [Milica Mihajlija](#), [Alan Kent](#), and Keith Gu.