**Four top tips for developers building location-based apps***By John Hoopes*

As the Internet continues to evolve and mature, so do development tools and the technologists who use them. But what does a more mature approach to application development look like today? How do application developers manage growing complexity, and deliver stunning experiences for their users?

Apps that use location data present extra challenges. Users expect seamless experiences with up-to-date, reliable data – a significant design and engineering challenge. To help developers, here are my four best practices for building location-based apps.

1. **Inject some API-ness**

Developers building any application or feature that requires the use of external or third-party data are often faced with the problem of having to download large datasets from which relevant data must be extracted. Geospatial information, for example, is often delivered in sizable datasets, which must be manually processed, stored, managed and updated when new updates are released by the provider. This technical overhead requires a significant investment of time on the side of a developer, making many potentially valuable datasets infeasible to use.

A much easier and more targeted solution is to consume the data that’s needed, when it’s needed, which is where APIs come into play. Not only are APIs often a more efficient way for users to consume data, they also tend to lower the total cost of building an app. The API provider hosts the systems that contain the relevant data and takes charge of the updates and management, freeing up time for developers to focus on other tasks. APIs also ensure that the most current data is being consumed, which can often define the entire proposition of an app, especially if it’s one that relies on accurate location information.

Mapping data visualisations for example often only fulfil their purpose if they reflect the most recent data. APIs can ease the burden on developers by offering reliable, well-maintained data sources.

1. **Validate before you integrate**

Application development today necessitates the use of a variety of datasets, many of which come in incompatible formats. In fact, the chances that all required datasets are interoperable is very low, which could mean a significant amount of heavy lifting at the data integration stage. When it comes to geospatial data on the web, for example, the de facto standard for vector features is [GeoJSON](https://geojson.org/). However, spatial data that developers will need might come as shapefiles, which are designed for GIS applications, and others, such as [Geographical Markup Language](https://en.wikipedia.org/wiki/Geography_Markup_Language" \l ":~:text=The%20Geography%20Markup%20Language%20(GML,geographic%20transactions%20on%20the%20Internet.) (GML), [KML](https://en.wikipedia.org/wiki/Keyhole_Markup_Language), [GPX](https://en.wikipedia.org/wiki/GPS_Exchange_Format), [polylines](https://developers.google.com/maps/documentation/utilities/polylinealgorithm), [vector tiles](https://www.ordnancesurvey.co.uk/blog/2020/07/the-benefits-of-vector-tiles/) and so on – each requiring special tools to work with.

Researching datasets, formats and libraries is not a controversial tip by any means, but it bears emphasising. Planning and validating use cases before breaking ground on code will save time, cost and sanity. After all, it’s easier to change a wire frame than to rewrite code – or worse, realise that you have chosen to incorporate a poorly supported Javascript library or an incomplete data source.

Every developer knows well the pain of having to solve a seemingly entirely unique problem – and the joy of finding a complex problem has already been solved. There are many active and thriving open source communities online that can provide support for spatial web developers. GeoJSON files can be easily and natively visualised with the popular Javascript mapping libraries, like [Leaflet](https://leafletjs.com/), [Mapbox GL JS](https://docs.mapbox.com/mapbox-gl-js/api/) and [OpenLayers](https://openlayers.org/) - there is a wealth of information online on doing so. Other formats do not have the same support, which may make the process of converting an incompatible format into one that can be easily integrated with the rest of an application’s tech stack more difficult. Tools like [mapshaper](https://mapshaper.org/), [QGIS](http://qgis.com/), [GDAL](https://gdal.org/), [Turf.js](http://turfjs.org/) and the [shapefile package](https://github.com/mbostock/shapefile) can help developers convert between formats, reproject coordinates, and performs spatial analytics and manipulation processes.

It’s a cliché, but working smarter, not harder, is what developers should strive for.

1. **Managing asynchronicity**

The web presents some interesting challenges for developers. For one, we don’t know how long some processes will take. For example, how quickly data is fetched from an API depends on bandwidth, the amount of data, the server, and so on. This is especially relevant in apps using spatial data as datasets are often fetched from external APIs, and are often sizable.

This is complicated by the fact that JavaScript code often relies on assets loaded earlier – to use a variable *x*, *x* has to be declared and assigned a value. When that assignment operation takes an indeterminate amount of time, how do you know when to proceed to computing *x + 1*?

Fortunately, the JavaScript community has designed a sophisticated suite of solutions to this problem. A promise holds the place of a value that is not yet known – the eventual result of an asynchronous operation. When the finishes the promise is resolved – or rejected. By chaining promises together, programmers can create programs that handle asynchronous operations efficiently and cleanly.

Building on promises, ECMAScript 2017’s [async / await syntax](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Async_await) makes “asynchronous code easier to write and to read afterwards”, enhancing the toolbelt devs can use to deal with these asynchronous operations.

To write performant apps, developers often need to fetch data from multiple sources, handle asynchronous operations, but not wait for one process to finish before starting another – that is, they need to [run operations concurrently](https://medium.com/ideas-at-igenius/some-beginner-tips-for-concurrency-with-async-await-and-promise-all-dc28b5a4411e). One tool for this is the Promise.all() method, which processes an array of promises, only resolving once all operations have finished.

Understanding tooling and techniques is therefore essential. When it comes to asynchronous data, for example, JavaScript has a lot of data management tools into the language itself, which can vastly reduce the potential complexity, improve performance, and result in better applications.

1. **Don’t neglect platforms**

A key challenge for designers and developers is creating a coherent experience across different platforms. Building an incredible data visualisation feature for a webpage might work well on a laptop screen, but how well does this translate when viewed on a smartphone? For B2B applications, use cases are generally geared toward users sat at a PC in an office. But increasingly, compatibility with portable devices, such as smartphones, is a requirement.

For GIS developers, making compelling and usable mapping data visualisations that work on mobile is particularly difficult. On a webpage, hover states allow the app designers to provide users with interactivity that is crucial to UX – like detailed contextual data of a geographical feature shown in a pop-up, triggered by the mouse hovering on a map feature. The same visualisation on a mobile necessitates a ‘click or nothing’ scenario, which vastly impacts user experience and functionality.

Of course, it’s not possible to ignore mobile users, so design thinking must inform the early stages of any application that requires compelling data visualisations. Sometimes – especially with mapping visualisations meant to support routing or wayfinding – a mobile-first approach should be taken. Either way, think through the user needs early, and solicit feedback through user testing often.

So, there you have it. Just a few tips to consider before embarking on your journey to creating a location-based application.