

UNIVERSITY OF GLOUCESTERSHIRE

Building scalable 'Progressive Web Apps' with comparable performance to that of native mobile applications.

James Dinan

DECLARATION

This Research Paper is a product of my own work and does not infringe the ethical principles set out in the University's Handbook for Research Ethics.

I agree that this research project may be available for reference via any and all media by any and all means now known or developed in the future at the discretion of the University.

A handwritten signature in black ink, appearing to read 'Dinan', written in a cursive style.

James Dinan

Date: 7th December 2016

Contents

1. Introduction	4
1.1 Research Questions	5
1.2 Objectives	5
2. Literature Review	6
2.1 Native vs. Web Applications	6
2.2 Progressive Web Apps	7
2.3 Web Application Frameworks	8
2.4 Impact of Application Performance	8
2.5 Performance Metrics	9
2.6 Performance Testing Tools	9
2.7 Optimising Web Application Performance	10
2.7.1 Non-blocking JavaScript	10
2.7.2 Minification and Tree-Shaking	11
2.7.3 CSS Optimisations	11
2.7.4 Web Workers	12
2.7.5 Server-Side Rendering	14
2.7.6 GraphQL	15
2.7.7 Conclusion on Optimisation Techniques	16
3. Methodology	17
3.1 Ethical Issues	19
4. Conclusion	20
5. References	21
6. Appendix 1: Requirements Specification & Wireframe	25

Figures

Figure 1 - Example of blocking JavaScript loaded in <head> tag	10
Figure 2 - Example of non-blocking JavaScript using the async property	11
Figure 3 - A JavaScript function before and after minification	11
Figure 4 - Combine opacity and pointer-events to optimise hiding elements	12
Figure 5 - Transforms and will-change to optimise animations	12
Figure 6 - UI thread creating and using data from a Web Worker.	13
Figure 7 - Worker.js: Worker posting message to UI thread after each computation	13
Figure 8 - Worker and UI thread communication diagram (Hwang, 2015, p.45)	14
Figure 9 - A GraphQL Query and Response	15

1. Introduction

Advances in browser features, new application programming interfaces (APIs) and changes in design principles have brought about the concept of Progressive Web Apps (PWAs). A PWA is much like an existing web application, built using current web development languages, HTML, CSS and JavaScript and hosted on a remote server. However, offline functionality, full-screen viewing, and the ability to pin the application to the home screen, such that it can be accessed in an identical manner to a natively installed application, are key characteristics behind a PWA (Malovolta, 2016, p.2). The intent is to develop applications that take advantage of the inherent benefits of the web platform, such as its portability, whilst delivering a comparable end user experience to that of native applications crafted specifically for mobile operating systems.

Native apps achieve higher levels of user engagement, with results from a recent survey comparing native and web use showing that users spend 87% of their time using native applications (ComScore, 2016, p.12). Traditionally there has been a perception that native apps have offered better performance through faster loading times, smoother animations with more consistent frame rates and reduced lag on user interactions. Web audiences, nevertheless, are "almost 3x the size" (ComScore, 2016, p.15), suggesting that web apps pose greater discoverability and a potential source of revenue for organisations, but are not offering a comparable alternative to native apps to sufficiently engage the user.

The aim of this research paper is to form a set of best practices that can be followed to optimise a PWA's performance, such that it offers a competitive alternative to existing native apps. This will be achieved through development of a weather forecasting PWA based on the functionality of the mobile website for the Met Office (Met Office, 2016a), the UK's National Meteorological Service, and testing the effect of optimisation techniques upon this. Whilst the techniques applied may boost performance, it will be significant to identify the cost, in terms of development complexity, that implementing these will cause (Malovolta, 2016, p.2). The balance between complexity and performance gains will be factored when forming these best practices to ensure solutions can be maintained and developed at scale within industry.

1.1 Research Questions

To guide in achieving the aim of this paper, a set of research questions were formulated addressing key areas:

- I. Which optimisation techniques can be applied to a PWA, and what effect do each of these have upon the performance and development complexity of the application?
- II. Can a PWA replicate the performance and end user experience of a native application?
- III. What are the best techniques for building a high performance PWA within industry, whilst ensuring the development is still scalable and maintainable?

1.2 Objectives

These research questions will be answered through the achievement of the following objectives:

- I. Develop a weather forecasting PWA and test whether potential optimisation techniques effect the performance of the app. Critique the development complexity of implementing each technique.
- II. Compare the performance and functionality of the developed PWA to the existing Met Office native application (Met Office, 2016b), and competitor native apps, through gathering user feedback on the end experience.
- III. Formulate a set of best practices, based on the collated findings on performance and complexity, that can be used to build scalable and optimised PWAs.

2. Literature Review

2.1 Native vs. Web Applications

Whilst native applications appear to be the more popular development solution, the web offers significant wider benefits that have brought about the want for an experience that emulates native. Cerf (2016) identifies one such characteristic difference, namely that web applications are more portable across the different mobile operating systems as there is a consistent interpretation of high-level languages within browsers. Native applications, compiled specifically for each platform are more efficient, but separate code bases are needed and must be "crafted to fit the operating system application programming interfaces and services available" (Cerf, 2016, p.7). This development for multiple platforms, often with multiple versions of each platform, is expensive and there is a trade between the optimisations that are achieved and the high development and maintainability costs (Wasserman, 2010, p.400).

As web applications are accessible via a browser they have a key benefit over native in that they are easily shareable via a URL. As suggested by Malavolta (2016, p.2), this additionally leads to no distribution delays when releasing software, as changes are instantly accessible. This benefit over native aligns closely with the transition in recent years from traditional process driven approaches to Agile methodologies, where "the development model relies on many successive releases of the evolving product" (Wasserman, 2010, p.398). In the case of the Android native app for the social network Twitter, it was found it takes up to 14 days for 75% of users to upgrade (Sillars, 2015, p.8), delaying distribution of the latest bug fixes and features, and limiting the effectiveness of any Agile approach to native development.

The web does lack support for some features that require access to device hardware; contacts, calendar and telephony data is not fully supported (Malavolta, 2016, p.2), but access to the camera, geolocation and push notifications, messages from the server that reengage the user, are achievable through web API's. The lack of installation is a benefit over native, saving on device storage space. A case study for a leading online real estate platform in India, Housing.com, indicates users were hesitant to download its native application on their low-end devices (Google, 2016b), and drove the company's move away from the platform. Despite the benefits web applications offer, Wasserman (2010, p.400)

points there has been uncertainty as to whether the experience and performance of such apps meets the needs of the market. Progressive Web Apps aim to address this issue, taking advantage of new technologies to deliver the best aspects of both web and native applications (Google, 2016a).

2.2 Progressive Web Apps

A core principle of PWA's is that of progressive enhancement; the adding of more features based on compatibility with the new technologies (Firtman, 2016, p.15). Chrome, Firefox and Opera browsers for the Android operating system are currently fully compatible (Firtman, 2016, p.16). Malovolta (2016, p.2) and Russell (2016) list three distinct conditions that must be true for an application to be recognized as a PWA:

- I. **HTTPS:** The application must be served via HTTPS (Hyper Text Transport Protocol Secure). HTTPS adds encryption which prevents intruders from tampering with any data and protects the security and privacy of the user (Grigorik, 2013a).
- II. **Web App Manifest:** There must be a defined web app manifest. This is a JSON-based file where metadata can be added that defines how an application opens, any icons for the mobile home screen, and enables the experience of the web app to be "more comparable to that of a native application" (W3C, 2016a).
- III. **Service Workers:** The application must use a Service Worker; a specialised script that runs in the background and enables control over how network requests within the application are handled (Gaunt, 2016). Service Workers allow programmatic caching or preloading of assets and data (Malovolta, 2016, p.2). To be considered a PWA, an application must use a Service Worker to load when there is no network connection (Gaunt, 2016).

Using an application shell architecture is suggested as a best approach when developing PWA's. This 'shell' defines minimal code required for the user interface, for example any navigation (Osmani and Gaunt, 2015). Caching this content via a Service Worker can allow this to load instantly and could be considered like the user interface code packaged within a native app. Any further content can be added progressively after this has rendered (Osmani and Gaunt, 2015). PWAs can be developed without a web framework, as the core technologies are independent of these, though frameworks have been used for web

applications to ease the development of dynamic content and simplify testing (Johanan, Khan and Zea, 2016, p.711).

2.3 Web Application Frameworks

Application frameworks, AngularJS, ReactJS, Backbone.js or Ember.js, are based around traditional software design patterns, which aim to provide reusable solutions to recurring software problems, typically the separation of concern between the user interface and application data (Syromiatnikov and Weyns, 2014, p.30). These frameworks follow patterns such as MVC (Model-View-Controller). This model is structured into 3 logic components; the model, which manages the system data, the view, which defines how this is presented to users, and the controller, which manages the user interactions (Sommerville, 2016, p.178). A challenge within software engineering is 'scaling-up', finding techniques that manage the project as complexity increases. Approaches suitable for individuals may not be suitable when development is split amongst a team on a large, complex project (Wasserman, 2010, p.398). The separation principle of the design patterns address this and allow work to be more easily distributed (Syromiatnikov and Weyns, 2014, p.22).

As of November 2016, usage statistics show the most popular of the 4 frameworks mentioned, AngularJS and Backbone.js, are used on a combined total of 1,066,412 websites globally (SimilarTech, 2016), proving their considerable user base. AngularJS is used on 5.2% of the top 10k most visited sites, making it the most popular of frameworks based on design patterns in this category (SimilarTech, 2016). When examining options for developing a PWA, it might be concluded that inclusion of frameworks, due to the increased resources to download, would negatively affect the initial load time. However, the framework structure could be considered an essential component to ensure maintainability when several developers work together or the application is sufficiently complex. This structure may also increase the development time that can be spent looking at optimisations; important given recent studies demonstrating the negative impact poor application performance has on user satisfaction.

2.4 Impact of Application Performance

Research by DoubleClick (2016, p.4) suggests up to "53% of visits are abandoned if a mobile site takes more than three seconds to load". This can negatively affect the revenue organisations generate through their website or application. Those that load within 5

seconds can double the money made through advertising compared to sites that only load within 19 seconds (DoubleClick, 2016, p.12). The speed at which actions execute within an app, not just initial load, are also of significance, with 49% of individuals in a research survey conducted by Hewlett Packard Enterprise (2016, p.4) indicating they expected a native app to perform actions within 2 seconds. 48% of individuals have uninstalled a native app that regularly ran slowly, while 33% stopped using the application entirely (Hewlett Packard Enterprise, 2016, p.5). This suggests PWA's, which to the end user will look and function as native apps, would be expected to meet similar targets, with loading, response and animation times matching the fastest native apps to compete. To that aim appropriate metrics to test against are essential.

2.5 Performance Metrics

RAIL, Response-Animation-Idle-Load, is a methodology that defines target metrics, based on user satisfaction, for distinctive life cycle events of a web application (Firtman, 2016, p.76). These are identified by Kearny (2016):

- I. **Response:** The time taken to respond to any user interactions should be less than 100ms. Interactions longer than 500ms should provide visual feedback.
- II. **Animation:** Animations, including scrolling, should render at a stable 60 frames per second; each frame rendering within 16ms.
- III. **Idle:** Defer nonessential execution of code to idle time when the user is not interacting. This should execute within 50ms to not block user interactions.
- IV. **Load:** Application loading should take less than 1000ms. Only critical content is essential, progressive rendering can be used for additional features.

As PWA's utilise Service Workers to quicken load times, Firtman (2016, p.77) suggests that focussing on improvements for response and animation times would be a recommended priority; though this point neglects data loading from external API's which is an additional important consideration.

2.6 Performance Testing Tools

Development tools available within browsers, such as Firefox and Chrome, enable testing of a PWA against the RAIL methodology. Measurements for frame rate, network requests, and function execution times are combined into reports to capture an in-depth analysis at any point in an applications lifecycle (Firtman, 2016, pp.80-81). These tools facilitate remote

debugging; the ability to test on mobile hardware (Firtman, 2016, p.83), and generate more realistic test results.

An additional tool of note is the Lighthouse browser extension that audits the application against the PWA core requirements, such as checking for a Service Worker (GoogleChrome, 2016). It also shows a 'speed index' metric that defines how much blank content is displayed during the application load (Firtman, 2016, pp.71-74), a significant metric to test given the RAIL requirement to load critical content within 1000ms. The tool is additionally available as a command line interface, which could allow the analysis to be run as part of any continuous integration and enable a more maintainable and efficient identification of any performance regression.

2.7 Optimising Web Application Performance

A contributing factor to the performance issues of web applications is that JavaScript is parsed and interpreted at runtime. Constructs like dynamic typing, where variable types are constantly validated (Dot, Martinez, and Gonzalez, 2015, p.41), limit its efficiency compared to native compiled languages (Oh and Moon, 2015, p.179). The requirement to download all resources also negatively impacts initial load. Despite restrictions, research has identified the following techniques that may optimise web application performance:

2.7.1 Non-blocking JavaScript

Firtman (2016, p.289) suggests a basic optimisation targeted towards the app's initial load; ensuring JavaScript loading does not happen within the head of an HTML file, figure 1. This prevents rendering of the page until these files are downloaded (Rajeev and Bakula, 2015, p.674). Moving scripts to the end, before the closing body tag, is suggested. Alternatively, the async property, figure 2, lets the script download without stopping rendering, allowing scripts within the head (Firtman, 2016, pp.166-167). These structural changes could improve the perceived load time of an app and are simple to implement.

```
<html>
  <head>
    <script src="blocking_javascript.js"></script>
  </head>
  <body>
    <h1>Web App Title</h1>
  </body>
</html>
```

Figure 1 - Example of blocking JavaScript loaded in <head> tag.

```

<html>
  <head>
    <script async src="non_blocking.js"></script>
  </head>
  <body>
    <h1>Web App Title</h1>
  </body>
</html>

```

Figure 2 - Example of non-blocking JavaScript using the async property.

2.7.2 Minification and Tree-Shaking

Minifying JavaScript and CSS, the removal of unnecessary characters from code (Rajeev and Bakula, 2015, p.672) is recommended as it offers file size reduction, lessening download times. Rajeev and Bakula (2015, p.672) demonstrate the application; more than halving the size of a JavaScript file. Minification is reasonably hassle-free to implement with popular online tools like Google's Closure Compiler (Firtman, 2015, p165). JavaScript code that follows the latest specification, ES2015, can see further reduction through tree-shaking with the rollup.js library (Rollup, 2016). ES2015 specification JavaScript uses import statements to load external functions; rollup.js can analyse the code base and create a more efficient bundle by only including functions that are referenced, eliminating redundant code (Rollup, 2016).

```

//original function
function add (number1, number2) {
  return number1 + number2;
}

//minified function
function add(a,b){return a+b}

```

Figure 3 - A JavaScript function before and after minification.

2.7.3 CSS Optimisations

Toggleing the visibility or display CSS property is often used to hide or show elements. This can negatively affect frame rate as a recalculation of the page layout and paint processing, the rendering of pixels by the browser, is required. A proposed solution is to combine pointer-events and opacity properties (Google Chrome Developers, 2016). Pointer-events control when an element is a target for click and touch events (Johanan, Khan and Zea, 2016, p.276). This allows for overlapping elements that do not block interactions of underlying layers. The opacity property sets the transparency of an element (Johanan, Khan and Zea, 2016, p.313). Combining these, figure 4, creates a hidden element that doesn't affect any interactions, but is not removed from the page layout. This leads to frame

rate improvements as the browser does not need to recalculate layout position when changing visibility (Google Chrome Developers, 2016).

```
nav {  
  opacity: 0;  
  pointer-events: none;  
}
```

Figure 4 - Combine opacity and pointer-events to optimise hiding elements

Size and positional properties, width or left, are used to animate elements, but these affect page layout and hinder performance. The transform property can also move elements, but is optimised by the browser. The element is isolated to a separate layer; only its position needs to be rendered between animation frames (Google Chrome Developers, 2016). However, a delay occurs if this isolation is done when starting an animation. Using the will-change property, figure 5, which defines properties to be modified in future, enables the browser to perform optimisations earlier (W3C, 2016b). Animating the position of an element using transform may require use of the scale property. This has a side-effect of scaling any child content. As such the architecture of HTML mark-up may need to change to use sibling elements, which could complicate development (Google Chrome Developers, 2016).

```
nav {  
  will-change: transform;  
  transition: transform 0.2s;  
  transform: scale(2)  
}
```

Figure 5 - Transforms and will-change to optimise animations

2.7.4 Web Workers

A constraint to frame rate and response with PWA's is the single-threaded execution model of JavaScript; all logic is executed sequentially on a single process responsible for the user interface, the UI thread. Any user interactions are handled, computations executed and then a response shown (Hwang, 2015, p.42). This single-threaded model prevents the power of multicore processors in modern smartphones from being taken advantage of (Wenzel and Meinel, 2015, p.140). The Web Worker API was introduced to allow scripts to execute in the background (W3C, 2015a). The RAIL methodology (Kearny, 2016), states individual frames have 16ms to render for a smooth frame rate. Functions that take too long to execute in the single-threaded model can stop frames from hitting this target. With the Web Worker API, computations can be run concurrently, separate from the UI thread, on a Worker thread. This multi-threaded approach means computations do not interfere with the

browser rendering the interface (Wenzel and Meinel, 2015, p.140). Figures 6 and 7 show how a Web Worker can be created and update the UI thread.

```
worker = new Worker("worker.js");
worker.onmessage = function(event) {
    document.getElementById("result").innerHTML = event.data;
};
```

Figure 6 - UI thread creating and using data from a Web Worker.

```
var _count = 0

function timer() {
    _count++;
    postMessage(_count);
    setTimeout("timer()", 1000);
}

timer();
```

Figure 7 - Worker.js: Worker posting message to UI thread after each computation.

Limitations apply in that communication between threads is only possible through message passing as no memory is shared. Web Workers also have no access to the DOM, the Document Object Model, (Wenzel and Meinel, 2015, p.140), the API used to manipulate HTML mark-up (W3C, 2015b). Figure 8 shows these communication restrictions. These limitations exist to ensure the code can be executed by multiple threads safely at the same time, but they complicate implementing Web Workers. Wenzel and Meinel (2015) have used Web Workers effectively for the parsing of data though did comment on the complications due to lack of DOM API access (Wenzel and Meinel, 2015, pp.141-142). Research has used Web Workers so computations can be shifted server-side and shared amongst devices (Hwang, 2015). Performance improvements were found to be reduced however by the latency, the time taken to send a packet of data (Grigorik, 2013b), between server and client (Hwang, 2015, p.45). This would seemingly worsen on slower mobile networks and suggests this particular use of Web Workers would not be effective for a PWA.

Work has been undertaken to simplify Web Workers as part of the AngularJS framework. It is possible for the core framework and business logic to be run in the background. Updating of the interface is then handled by Angular API's (AngularConnect, 2015). The framework's MVC architecture and ability to handle DOM manipulation through data binding, whereby any changes of the data are updated on the view automatically, may remove any need for DOM access in worker threads and negate some complications, allowing PWA's to leverage multi-threading like native apps for improved frame rates and responsiveness.

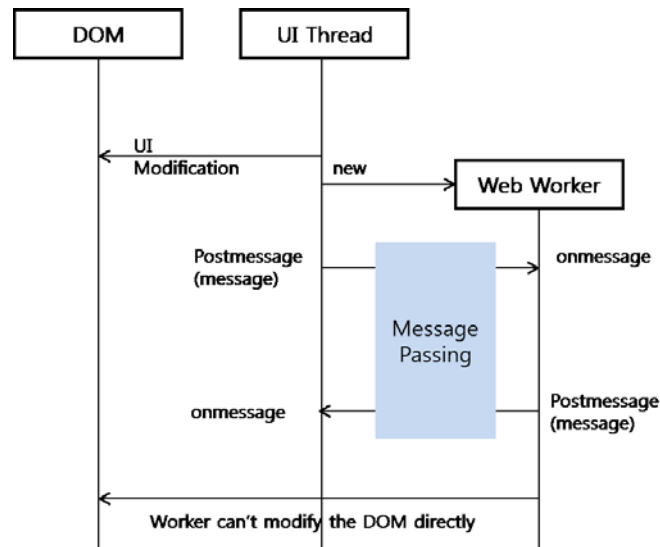


Figure 8 - Worker and UI thread communication diagram (Hwang, 2015, p.45)

2.7.5 Server-Side Rendering

The application frameworks create single page applications (SPA's) where the server sends an initial HTML file and payload of assets, images and JavaScript files, and then the client takes control. This improves performance when changing pages as a complete HTML document does not have to be fetched and parsed (Strimpel and Najim, 2016, p.9). A test of hybrid applications found HTML rendering to be even faster than the lifecycle management inside a native application when switching views (Willocx, Vossaert and Naessens, 2016, p.43), suggesting PWA's may even hold an advantage over native. A weakness is waiting for the initial download of JavaScript files that form frameworks (Strimpel and Najim, 2016, p.3). No interactions are possible until these load.

Isomorphic JavaScript, the sharing of the "same JavaScript code between the browser client and the web application server" (Strimpel and Najim, 2016, p.4), is as an approach that can address these slow application load times. Several solutions exist for the popular JavaScript frameworks to facilitate the rendering of the initial view on the server. Angular Universal (Google, 2016c) and Rendr for Backbone.js (Rendrjs, 2016) are such examples. In the case of Angular Universal, the server rendered view that is loaded can record any user events and make calls to retrieve data. These are then able to be seamlessly passed to the client version once this has loaded (NG-Conf, 2016).

2.7.6 GraphQL

GraphQL is a query language that allows the client application to determine which data is returned from a server (Facebook, 2016). The traditional approach is to script multiple server-side endpoints; URL's that when accessed return a subset of data, typically based on the popularised REST, or Representational State Transfer, architecture (Sommerville, 2016, p.530). When several resources are needed a REST approach can result in multiple requests or unnecessary data transfer, causing suboptimal performance (Cederlund, 2016, pp.10-11). As GraphQL grants control over to the client, only a single endpoint is required and any redundant information can be removed. Figure 9 shows a GraphQL request and response, where a request is made for a user with id 001 to obtain name and gender properties. The query filters to the user and only returns the specified fields, nothing more.

Cederlund (2016) has analysed the performance effect of GraphQL and the findings suggest it results in decreased response time where achieving the same end data would require a combination of REST API requests. Where only a single request would be required however, a regular REST endpoint has better response times (Cederlund, 2016, p.82). This research is limited in that it was conducted on a wireless connection, not a slower mobile network. Testing will be required to conclude if any performance gains are still offered in such a case.

```
GraphQL query selecting user with name and gender:
{
  user(id: "001") {
    name
    gender
  }
}

Data returned from query matches passed in structure:
{
  "data": {
    "user": {
      "name": "Joe Bloggs",
      "gender": "Male"
    }
  }
}
```

Figure 9 - A GraphQL Query and Response

2.7.7 Conclusion on Optimisation Techniques

The removal of render blocking JavaScript and file size reduction through minification and tree-shaking appear the least complex of the optimisations. Tree-shaking does require using the newer syntax of JavaScript, reducing its use retrospectively, though this research paper is addressed primarily with new development. The other techniques all offer potential performance gains that could offer PWA's greater parity with native, though have inherent complications through necessary architectural changes. Server-side rendering is additionally only possible using a server capable of rendering JavaScript, typically Node.js (Strimpel and Najim, 2016, p.13), something that may not be maintainable by all organisations. These complications could prove stumbling blocks when developing an optimised PWA at scale within industry and therefore will be evaluated further through primary research.

3. Methodology

A mixed method approach using both quantitative and qualitative research methods (Creswell, 2013, p.4) will be used for the collection of primary data to provide a more complete answer to the aim of this paper. This will follow a sequential explanatory design strategy (Creswell, 2013, p.15), whereby initial quantitative tests of the optimisation techniques will then be followed by qualitative research drawing upon complexity and how the end experience compares to native. The following methods address each objective:

- I. A PWA will be developed that partially mirrors the functionality of the Met Office mobile website (Met Office, 2016a). A specification and wireframe is included in appendix 1. The application will be audited using the Lighthouse extension to ensure it meets the minimum technical requirements of a PWA. The identified optimisation techniques will then be applied and quantitative test data collected for the following performance parameters addressing metrics of the RAIL methodology:
 - Application load time
 - Speed index
 - Time to first meaningful paint (time application is interactive)
 - Frame rate
 - Response time to user actions
 - Resource loading time

The first three tests will be taken on both initial and repeat visits. To ensure the integrity and reliability of all tests, they will be evaluated using Chrome developer tools, with remote debugging on both a high and low end Android device, table 1. Connections will be throttled to allow testing on stable wireless and 3G network connections. The time to first meaningful paint and speed index metrics will be additionally assessed with the Lighthouse extension.

The PWA and test results will be presented at a development seminar held at the Met Office. Qualitative data will then be collected anonymously through unstructured interviews (Cohen and Crabtree, 2006) with software development professionals. These interviews will gauge opinions on any complications towards developing the PWA within industry and the individual complexity of each optimisation. Unstructured

interviews allow prior views on complexity to be developed upon and revised by the respondents.

	Low End	High End
Vendor	Sony	Samsung
Device	Xperia E1	Galaxy S6
RAM	512MB	3GB
Processor	Qualcomm MSM8210	Exynos 7420 Octa
CPU Frequency	Dual-core 2 x 1.2 GHz	Octa-core 4 x 2.1 GHz 4 x 1.5 GHz
Internal Memory	4GB	64GB
Operating System	Android 4.4.2 (KitKat)	Android 6.0.1 (Marshmallow)
Browser	Google Chrome (version 54)	Google Chrome (version 54)

Table 1 - Specification of low and high end Android devices to be used for testing

- II. Feedback will be gathered on the end user experience of the PWA and how this compares to native applications through an online questionnaire. This was decided instead of comparative testing of native apps due to the differing metrics produced by web and native testing tools. Participants will be selected from Met Office employees and University of Gloucestershire students; with a link to the questionnaire distributed by email and through the social platform LinkedIn. This will contain a mix of closed and open questions.

Users will be instructed to perform actions with the developed PWA and rate this to their experience using the Met Office and competitor native applications. Open questions will collect qualitative data; allowing participants to expand upon answers. This will draw insights towards the PWA's performance not covered by the research. The questions that form this questionnaire are dependent upon the PWA and will be formulated once this has been developed.

- III. Quantitative data from the optimisation tests and questionnaire comparisons will be graphically represented, evaluated and form the starting point of any recommendations. The interviews on complexity and the qualitative questionnaire results on performance will then be analysed to identify any recurring trends or bias.

A final set of best practices combining analysis from the mixed research methods will then be produced aimed at building PWA's with native performance.

3.1 Ethical Issues

Participants in the unstructured interviews and questionnaire will be made aware that all responses will be anonymous. Clear indication of the purpose of the study will be given and any participation will be voluntary and can be withdrawn. All testing and analysis of data will be conducted in an ethical manner, using licensed versions of software.

4. Conclusion

It is identifiable from the research that the web platform offers substantial benefits over native to organisations and end users. Costs can be reduced, less code is needed to be maintained and updates are more easily distributed. The aim of PWA's is to provide a web experience that emulates the look and appearance of native apps. Opening in full screen from a clickable icon makes it impossible for end users to differentiate between the two. Any applications in their eyes would be expected to perform equally. The negative effect of poor performance on user satisfaction is clearly identified by the research. For PWA's to be a viable commercial alternative they must overcome the inherent performance issues associated with the web platform, something which the primary research of this paper will aim to solve.

5. References

- 1) AngularConnect. (2015). *Using Web Workers for more responsive apps – Jason Teplitz*. Available at: https://www.youtube.com/watch?v=Kz_zKXiNGSE (Accessed: November: 2016)
- 2) Cederlund, M. (2016). *Performance of frameworks for declarative data fetching: An evaluation of Falcor and Relay+GraphQL*. Master's Thesis. KTH Royal Institute of Technology, Stockholm, Sweden. Available at: <http://kth.diva-portal.org/smash/get/diva2:1045900/FULLTEXT01.pdf> (Accessed: November 2016)
- 3) Cerf, V.G. (2016). 'Apps and the web', *Communications of the ACM*. 59(2), p.7. doi: 10.1145/2872420
- 4) Cohen D. and Crabtree B. (2006). *Qualitative Research Guidelines Project*. Available at: <http://www.qualres.org/HomeUnst-3630.html> (Accessed: December 2016)
- 5) ComScore. (2016). *The 2016 U.S. Mobile App Report*. Available at: <http://www.comscore.com/Insights/Presentations-and-Whitepapers/2016/The-2016-US-Mobile-App-Report> (Accessed: October 2016)
- 6) Creswell J. W. (2013). *Research Design: Qualitative, Quantitative and Mixed Methods Approaches*. 4th edn. London: SAGE Publications
- 7) Dot, G., Martinez, A. and Gonzalez, A. (2015). 'Analysis and Optimization of Engines for Dynamically Typed Languages'. *2015 27th International Symposium on Computer Architecture & High Performance Computing (SBAC-PAD)*, IEEE, pp41-48
- 8) DoubleClick. (2016). *The need for mobile speed. Better user experiences, greater publisher revenue*. Available at: <https://www.doubleclickbygoogle.com/articles/mobile-speed-matters/> (Accessed: October 2016)
- 9) Facebook (2016). *GraphQL: Working Draft – October 2016*. Available at: <http://facebook.github.io/graphql/> (Accessed: November 2016)
- 10) Flirtman, M. (2016). *High performance mobile web: Best practices for optimizing mobile web apps*. Sebastopol, California: O'Reilly Media, Inc.
- 11) Gaunt, M. (2016). *Service Workers: an Introduction*. Available at: <https://developers.google.com/web/fundamentals/getting-started/primers/service-workers> (Accessed: November 2016).

- 12)Google. (2016a). *Progressive Web Apps*. Available at:
<https://developers.google.com/web/progressive-web-apps/> (Accessed: October 2016)
- 13)Google. (2016b). *Housing.com increases conversions and lowers bounce rate by 40% with new PWA*. Available at:
<https://developers.google.com/web/showcase/2016/pdfs/housing.pdf> (Accessed: October 2016)
- 14)Google. (2016c). *Angular Universal*. Available at: <https://universal.angular.io/> (Accessed: December 2016)
- 15)GoogleChrome (2016). *Lighthouse – Auditing, performance metrics, and best practices for Progressive Web Apps*. Available at: <https://github.com/GoogleChrome/lighthouse> (Accessed: December 2016)
- 16)Google Chrome Developers (2016). *High performance web user interfaces - Google I/O 2016*. Available at: <https://www.youtube.com/watch?v=thNyy5eYfbc> (Accessed: November 2016)
- 17)Grigorik, I. (2013a). *High Performance Browser Networking – Transport Layer Security (TLS)*. Available at: <https://hpbnp.co/transport-layer-security-tls/#https-everywhere> (Accessed: November 2016)
- 18)Grigorik, I. (2013b). *High Performance Browser Networking – Primer on Latency and Bandwidth*. Available at: <https://hpbnp.co/primer-on-latency-and-bandwidth/> (Accessed: November 2016)
- 19)Hewlett Packard Enterprise. (2015). *Failing to Meet Mobile App User Expectations: A Mobile User Survey - Dimensional Research*. Available at:
<http://go.saas.hp.com/techbeacon/apppulse-mobile-survey> (Accessed: October 2016)
- 20)Hwang, I. (2015). 'Design and implementation of cloud offloading framework among devices for web applications', *2015 12th Annual IEEE Consumer Communications & Networking Conference (CCNC)*, IEEE, pp.41-46
- 21)Johanan, J., Khan, T. and Zea, R. (2016). *Web Developer's Reference Guide*. Birmingham, UK: Packt Publishing Ltd.
- 22)Kearny, M. (2016). *Measure Performance with the RAIL Model*. Available at:
<https://developers.google.com/web/fundamentals/performance/rail> (Accessed October 2016).

- 23)Malavolta, I. (2016). 'Beyond native apps: web technologies to the rescue! (keynote)', *In Proceedings of the 1st International Workshop on Mobile Development*, ACM, New York, NY, USA, 1-2
- 24)Met Office (2016a). *Met Office Mobile Weather*. Available at: <http://www.metoffice.gov.uk/mobile/> (Accessed: November 2016)
- 25)Met Office (2016b). *Met Office Weather App*. Available at: <http://www.metoffice.gov.uk/services/mobile-digital-services/weather-app> (Accessed: November 2016)
- 26)NG-Conf. (2016). *Angular 2 Universal Patterns - Jeffrey Whelpey, Patrick Stapleton*. Available at: https://www.youtube.com/watch?v=TCj_oC3m6_U (Accessed: November 2016)
- 27)Oh, J. and Moon, S. (2015). 'Snapshot-based loading-time acceleration for web applications', *2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, IEEE, San Francisco, CA, 2015, pp. 179-189.
- 28)Osmani, A. and Gaunt, M. (2015). *Instant Loading Web Apps with an Application Shell Architecture*. Available at: <https://developers.google.com/web/updates/2015/11/app-shell> (Accessed October 2016)
- 29)Rajeev B.V. and Bakula, K. (2015). 'A developer's insights into performance optimizations for mobile web apps', *Advance Computing Conference (IACC), 2015 IEEE International*, Bangalore, 2015, pp. 671-675. doi: 10.1109/IADCC.2015.7154791
- 30)Rendrjs. (2016). *Rendr*. Available at: <https://github.com/renderjs/render> (Accessed: December 2016)
- 31)Russell, A. (2016). *What, Exactly, Makes Something A Progressive Web App?* Available at: <https://infrequently.org/2016/09/what-exactly-makes-something-a-progressive-web-app/> (Accessed: November 2016)
- 32)Sillars, D. (2015). *High performance Android apps: Improve ratings with speed, optimizations, and testing*. Sebastopol, California: O'Reilly Media, Inc.
- 33)SimilarTech. (2016). *Technologies Market Share - JavaScript*. Available at: <https://www.similartech.com/categories/javascript> (Accessed: November 2016)

- 34) Sommerville, I. (2016). *Software Engineering*. 10th edn. London: Pearson Education Limited.
- 35) Strimpel, J and Najim, M. (2016). *Building Isomorphic JavaScript Apps: From concept to implementation to real-world solutions*. Sebastopol, California: O'Reilly Media, Inc
- 36) Syromiatnikov, A and Weyns, D. (2014). 'A Journey through the Land of Model-View-Design Patterns', *2014 IEEE/IFIP Conference on Software Architecture (WICSA)*, IEEE, Sydney, NSW, 2014, pp. 21-30
- 37) Wasserman, A. I. (2010). 'Software engineering issues for mobile application development', *In Proceedings of the FSE/SDP workshop on Future of software engineering research*, ACM, New York, NY, USA, pp.397-400
- 38) Wenzel, M and Meinel, C. (2015) 'Parallel network data processing in client side JavaScript applications,' *2015 International Conference on Collaboration Technologies and Systems (CTS)*, Atlanta, GA, 2015. IEEE, pp.140-147. doi: 10.1109/CTS.2015.7210414
- 39) Willocx, M., Vossaert, J. and Naessens, V. (2016). 'Comparing performance parameters of mobile app development strategies', *In Proceedings of the International Conference on Mobile Software Engineering and Systems*, ACM, New York, NY, USA, pp.38-47
- 40) W3C (2015a). *Web Workers - W3C Working Draft 24 September 2015*. Available at: <https://www.w3.org/TR/workers/> (Accessed: November 2016)
- 41) W3C (2015b). *W3C DOM4 - W3C Recommendation 19 November 2015*. Available at: <https://www.w3.org/TR/dom/> (Accessed: November 2016)
- 42) W3C (2016a). *Web App Manifest - W3C Working Draft 07 November*. Available at: <https://www.w3.org/TR/2016/WD-appmanifest-20161107/> (Accessed: November 2016)
- 43) W3C (2016b). *CSS Will Change Module Level 1*. Available at: <https://www.w3.org/TR/css-will-change/> (Accessed: November 2016)

6. Appendix 1: Requirements Specification & Wireframe

Purpose

Production of a Progressive Web Application (PWA) that offers forecast weather data targeted at the public. Type of data offered to mirror that currently delivered by the Met Office mobile website (<http://www.metoffice.gov.uk/mobile/>) and native application. The purpose of the application is to test the ability of PWA's to offer a comparable alternative to native platforms.

System Overview

Weather Data for the application will be acquired using the Met Office public API DataPoint (<http://www.metoffice.gov.uk/datapoint/product>). The application will make use of a text-based search functionality to provide weather forecasts for a specific location. This will be presented in the form of a table showing weather forecast information at 3 hourly intervals for the next 5 days, as well as summary text from forecasters. The AngularJS framework will be used for maintainability; ensuring the application will have the architecture to grow and develop to the scale required in industry. AngularJS was chosen due to being the current leader of MVC frameworks within the top 10k most visited sites.

The application will include the following key concepts of PWA's:

- Service Worker for offline functionality and caching.
- Application shell architecture.
- Home screen icon, full-screen viewing and splash-screen.
- Push notifications.

The following areas will be investigated for speed improvements after development:

- Async loading.
- Minification and Tree Shaking.
- CSS based performance improvements (pointer-events / opacity, transforms / will-change).
- Web Workers and separate UI Thread.
- Server-side rendering.
- GraphQL for the data service

Specific Requirements

Application to be hosted on a secure HTTPS domain as part of PWA requirements. This will be achieved through the free tier of Amazon Web Service (<https://aws.amazon.com/>).

Project Timetable

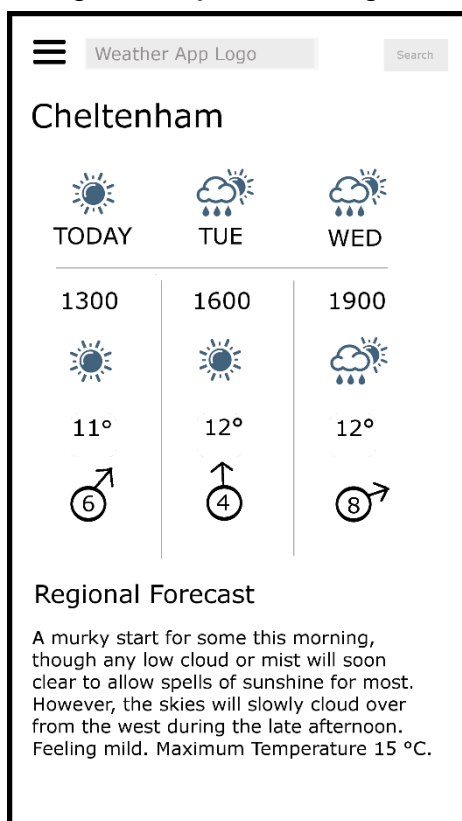
Estimated hours to complete initial application (minus optimisations):

Basic Structure of Application	10
Service to pull weather data	10
Displaying data in table format	7
Text Search	7
Add to home screen install banner, splash screen and icon	4
Service worker and basic offline caching	4
Push notifications	6
Total:	48

Proposing to build the initial PWA by January 2017. Demo of the product and test results to be given at a seminar at Met Office HQ, Exeter in late February / early March 2017.

Draft Wireframe

A preliminary wireframe has been produced showing the proposed look of the application, though is subject to change once development begins:



A search button will make a text input visible & animate from the top of the screen, where a new location can be entered and searched for. An additional burger menu animates a side navigation bar that will contain recently searched for destinations.

The top row of the forecast table gives an overview of the weather for the next 5 days (can be scrolled horizontally). Detailed information is then displayed showing weather type, temperature and a wind speed/direction icon for each of the 3-hourly time-steps. Regional forecast text will also be displayed beneath the forecast table.

UNIVERSITY OF GLOUCESTERSHIRE

Building scalable 'Progressive Web Apps' with comparable performance to that of native mobile applications.

James Dinan

DECLARATION

This Research Paper is a product of my own work and does not infringe the ethical principles set out in the University's Handbook for Research Ethics.

I agree that this research project may be available for reference via any and all media by any and all means now known or developed in the future at the discretion of the University.

A handwritten signature in black ink, appearing to read 'Dinan', is written above the printed name.

James Dinan

Date: 7th April 2016

Abstract

Whilst native applications currently appear to be the more popular development solution for mobile devices, the web platform offers significant wider benefits; increased portability, reduced costs, and faster distribution. Progressive Web Apps (PWAs) aim to provide a web experience that emulates the look and appearance of native apps whilst retaining these benefits. For PWA's to be a viable commercial alternative they must overcome the inherent performance issues associated with the web and be scalable; fitting with the requirements for maintainable code and having all the features deemed necessary by end users. A weather forecasting PWA was developed. Optimisations were tested and end user feedback comparing the applications' performance to existing native apps was gathered from 31 individuals. An interview was conducted with 5 software development professionals from the Met Office gauging views on complexity and viability of the approach. The findings have been used to formulate a set of best practice recommendations for the development of performant, scalable PWAs within industry.

Contents

1. Introduction	7
2. Design of PWA Weather	8
2.1 User Interface Design.....	8
2.2 Technical Design	9
3. Implementing PWA Weather and Optimisations	10
3.1 Optimisation Results and Discussion	11
3.1.1 Non-blocking JavaScript	12
3.1.2 Minification	15
3.1.3 Tree-Shaking	17
3.1.4 CSS: transform and will-change attributes	17
3.1.5 CSS: pointer-events and opacity attributes	19
3.1.6 Web Workers	21
3.1.7 GraphQL	22
4. Questionnaire.....	26
4.1 Design	26
4.2 Results and Discussion	27
4.2.1 Quantitative Data	27
4.2.2 Qualitative Data	33
5. Seminar and Interview Discussion	36
6. Best practice recommendations.....	42
7. Conclusion	44
7.1 Limitations and future work.....	44
8. References.....	45
9. Appendices	50
9.1 Appendix 1: Assignment 1 Feedback	50
9.2 Appendix 2: Response to Assignment 1 Feedback and Conclusion/Objectives	52
9.3 Appendix 3: Requirements Specification & Wireframe	54
9.4 Appendix 4: Lighthouse PWA Report for pwaweather.xyz (Dev branch).....	56
9.5 Appendix 5: Questionnaire	60
9.6 Appendix 6: Questionnaire Results	63
9.7 Appendix 7: Met Office Interview Transcript	68
9.8 Appendix 8: Met Office Seminar Presentation Slides	76
9.9 Appendix 9: PWA Weather Code (Dev branch).....	83
9.10 Appendix 10: CSS Non-Optimised Branch.....	127
9.11 Appendix 11: Web Worker Branch	128

9.12	Appendix 12: GraphQL Branch	131
------	-----------------------------------	-----

Figures

Figure 1 - PWA Weather final coded implementation; shows the forecast table and text forecast for Cheltenham, UK.	8
Figure 2 – Animated side navigation panel listing previous locations and search panel with input filter. Used to test CSS optimisations.....	9
Figure 3 - 'Add to Home Screen' browser prompt on Android Chrome triggered by implementing the web manifest and service worker.	10
Figure 4 - Chrome Developer Tools: how load tests were measured	12
Figure 5 - Load statistics for blocking/non-blocking JS on a Low-End device.....	13
Figure 6 - Load statistics for blocking/non-blocking JS on a High-End device.....	14
Figure 7 - Load statistics for minified/non-minified JS on a Low-End device.	16
Figure 8 - Load statistics for minified/non-minified JS on a High-End device.	16
Figure 9 - Chrome Developer Tools: how framerate was measured	18
Figure 10 - Framerate statistics for transform/will-change and positioning CSS.....	19
Figure 11 - Framerate statistics for pointer-events/opacity and visibility CSS	20
Figure 12 - Framerate statistics for web worker / no web worker	22
Figure 13 - Load statistics for GraphQL/REST API on a Low-End device.	24
Figure 14 - Load statistics for GraphQL/REST API on a High-End device.	25
Figure 15 - Frequently used weather applications of the questionnaire participants.	27
Figure 16 - Digital skill level of the questionnaire participants.	28
Figure 17 - User responses comparing the load speed of PWA Weather to a frequently used weather application.....	29
Figure 18 - Responses from all users comparing the load speed of PWA Weather grouped by Android and iOS.	29
Figure 19 - User responses comparing the forecast load speed of PWA Weather to a frequently used weather application.	30
Figure 20 - Responses from all users comparing the forecast speed of PWA Weather grouped by Android and iOS.	31
Figure 21 - User responses comparing the animation performance of PWA Weather to a frequently used weather application.	32
Figure 22 - Responses from all users comparing the performance of animations within PWA Weather grouped by Android and iOS.....	32

Tables

Table 1 - Metrics for testing application load (Firtman, 2016, p.65),	11
Table 2 - Test results for non-blocking JavaScript.....	12
Table 3 - Test results for blocking JavaScript.....	13
Table 4 - File sizes for minified and non-minified code.....	15
Table 5 - Test results for non-minified JavaScript files	15
Table 6 – File sizes for tree-shaked and non-tree-shaked code	17
Table 7 - Test results for transform/will-change attributes	18
Table 8 - Test results using CSS left positioning	18
Table 9 - Test results for pointer-events/opacity.....	20
Table 10 - Test results for CSS visibility attribute	20
Table 11 - Test results for search typing with Web Worker	21
Table 12 - Test results for search typing without Web Worker	21
Table 13 - GraphQL Query and Response for data from the Met Office DataPoint API.	23
Table 14 - Test results using the GraphQL service.	24
Table 15 - Test results using the REST API	24
Table 16 – Rationale for main performance questions of the end user questionnaire.	26
Table 17 - User responses to question “If you have any other comments about the performance of PWA Weather, please provide them below”.	33
Table 18 - Discussion of qualitative questionnaire responses	35
Table 19 - Discussion of points raised during Met Office interview.....	41

1. Introduction

Secondary research in the first half of this paper has shown that the web platform offers substantial benefits over native; costs can be reduced, less code is needed to be maintained and updates are more easily distributed. The negative effect of poor performance on user satisfaction however is clearly identified, and for Progressive Web Apps (PWAs) to be a viable commercial alternative they must overcome the inherent performance issues. This paper documents the development of a weather forecasting PWA. The intent was to create an app, PWA Weather, which could serve as a minimal viable product within industry, replicating the core functionality of the Met Office (Met Office, 2016b) and other competitor native apps.

PWA Weather was used as a platform to test optimisation techniques identified through secondary research. Following implementation, a questionnaire was used to collect data on its performance compared to existing native applications. The app and test results were presented at a seminar at the Met Office, and an unstructured interview undertaken with software professionals to discover opinions on scalability and complications towards development within industry. This data has been documented, discussed in context with secondary research, and used in conjunction with the optimisation results to form the basis of best practice recommendations. The first half of this paper defined the objectives below:

- I. Develop a weather forecasting PWA and test whether potential optimisation techniques effect the performance of the app. Critique the development complexity of implementing each technique.
- II. Compare the performance and functionality of the developed PWA to the existing Met Office native application (Met Office, 2016b), and competitor native apps, through gathering user feedback on the end experience.
- III. Formulate a set of best practices, based on the collated findings on performance and complexity that can be used to build scalable and optimised PWAs.

The following sections detail design and implementation of PWA Weather, and then a discussion of the optimisation test results. This is followed by sections on the questionnaire and interview; which discuss the results of these research methods and the implications of the findings. Finally, the set of best practice recommendations are discussed in Section 6.

2. Design of PWA Weather

2.1 User Interface Design

A requirements specification, as detailed in Appendix 3 (Section 12.3), includes a wireframe of the initial design for the user interface (UI) of the application. Changes in design were undertaken during development in consideration of accurately testing all optimisation techniques. A screenshot of the final application is shown in figure 1. A forecast table and summary text forecast information was included to present a comparable feature set to the existing Met Office mobile website (Met Office, 2016a), native app (Met Office, 2016b) and competitors; facilitating a more accurate comparison.

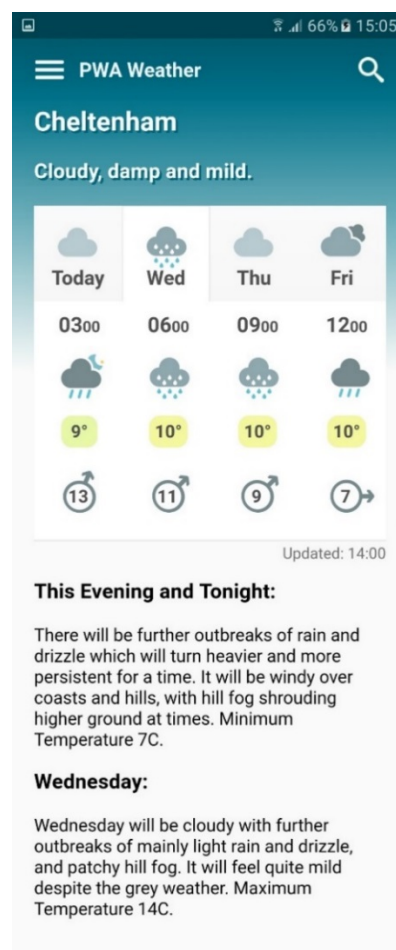


Figure 1 - PWA Weather final coded implementation; shows the forecast table and text forecast for Cheltenham, UK.

The creation of the forecast table was reliant upon multiple dependant calls to the Met Office DataPoint API (Met Office, 2017) and was included to act as a suitable test of GraphQL, which Cederlund (2016, p.82) suggested could offer performance gains. The side navigation, figure 2, was animated to appear from the left and was included to test the performance of CSS transforms and will-change attributes (Google Chrome Developers,

2016). A search panel showing a filter of locations was placed over the top of the main application page, figure 2, allowing the combination of pointer-events and opacity properties (Google Chrome Developers, 2016) to be tested.

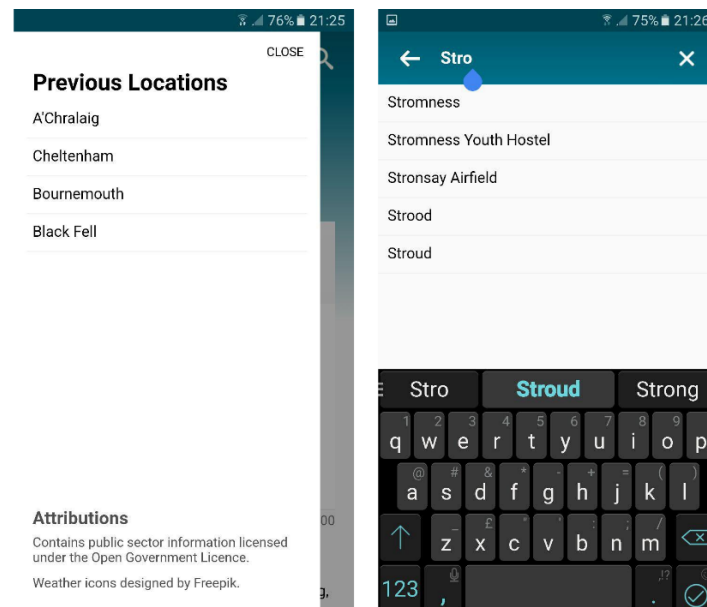


Figure 2 – Animated side navigation panel listing previous locations and search panel with input filter. Used to test CSS optimisations.

2.2 Technical Design

Technical design of PWA Weather considered the core requirements of a PWA; inclusion of a web app manifest, service worker and HTTPS (Russell, 2016), as well as scalability. Research in the literature review identified the Model-View-Controller (MVC) structure enforced by web frameworks an essential component to maintainability (Syromiatnikov and Weyns, 2014, p.22). AngularJS was shown to be the most popular framework (SimilarTech, 2016) and was deemed appropriate for PWA Weather. The AngularJS framework, version 2.4.4, was chosen for its stability. This was not updated to ensure a consistent code base and integrity of the tests.

A modular AngularJS component structure was designed for the home page of the application and the forecast table. This was chosen to allow the logic for each to be isolated. Services were designed for pulling in 3-Hourly forecast data, daily summary text, and forecast locations from the Met Office Datapoint API (Met Office, 2017). This API allows access by third parties to the Met Office's Public Sector information. Separation of data services and components was designed as it facilitates easier unit testing, important for scalability.

3. Implementing PWA Weather and Optimisations

The full code base of the develop branch can be seen in Appendix 9 (Section 9.9), with additions for optimisation tests provided in Appendices 10 to 12 (Sections 9.10 to 9.12). Implementation of PWA Weather began with the creation of the application shell defining core navigation and styling. This included the header, icons and animation of the side navigation and search results panel. A script was created for the minification of files and bundling of assets into a production build. This task included injecting the core CSS into the head of the index file to limit external requests.

The angular components and services were developed, pulling data from the API. The HTML mark-up for the forecast table was created by iterating over the data and binding this within table rows. This binding allows for the view to be updated by modifying the data model. A single sprite sheet was created for forecast icons to reduce HTTP requests. Icons were acquired from a free icon set (Freepix, 2017), and adapted to fit all weather types returned by the API. Wind directional icons were created using a vector editor and rotated to match all compass points. Allowing angular components nested within the application shell to communicate was a difficulty solved by the creation of a shared, singleton service to pass the state between the different components.



Figure 3 - 'Add to Home Screen' browser prompt on Android Chrome triggered by implementing the web manifest and service worker.

The web manifest and service worker scripts were added to the application root. These trigger the display of the 'Add to home screen' prompt on the Android Chrome browser, figure 3. The domain name pwaweather.xyz was purchased. Amazon Web Services (Amazon Web Services, 2017) was then chosen as a suitable location for hosting under this domain for end user testing. This offered a stable and performant server, and also free security certificates for the HTTPS connection necessary of a PWA (Russell, 2016). The final code was validated with Lighthouse (Firtman, 2016, pp.71-74), to ensure conformance with the PWA requirements, Appendix 4 (Section 9.4).

The following optimisations identified in the literature review were applied to PWA Weather and tested for performance improvements:

- Non-blocking JavaScript
- Minification
- Tree-shaking
- CSS transform & will-change attributes
- CSS pointer-events and opacity attributes
- Web Workers
- GraphQL

3.1 Optimisation Results and Discussion

To ensure integrity and reliability of all tests, they have been evaluated using Chrome Developer Tools, with remote debugging on both a high and low end Android device (see methodology for device statistics). Tests reliant on the network have been throttled to run on stable wireless and 3G connections. Testing of the metrics in table 1 has been performed to assess areas of the application load.

Metric	Explanation
First Paint	The time taken for the first page content to be visible.
First Meaningful Paint	First appearance of meaningful content to the user. In the case of PWA Weather this was defined as the appearance of the forecast text.
DOMContentLoaded	The time taken for the DOM to be fully constructed.
Load	The time taken for the DOM to be constructed and additional resources (images, styles) to be loaded.

Table 1 - Metrics for testing application load (Firtman, 2016, p.65),

DOMContentLoaded and the Load properties were available directly under the network tab of the Chrome Developer tools. The First Paint and Meaningful Paint metrics were assessed using the screenshot timeline, see figure 4.

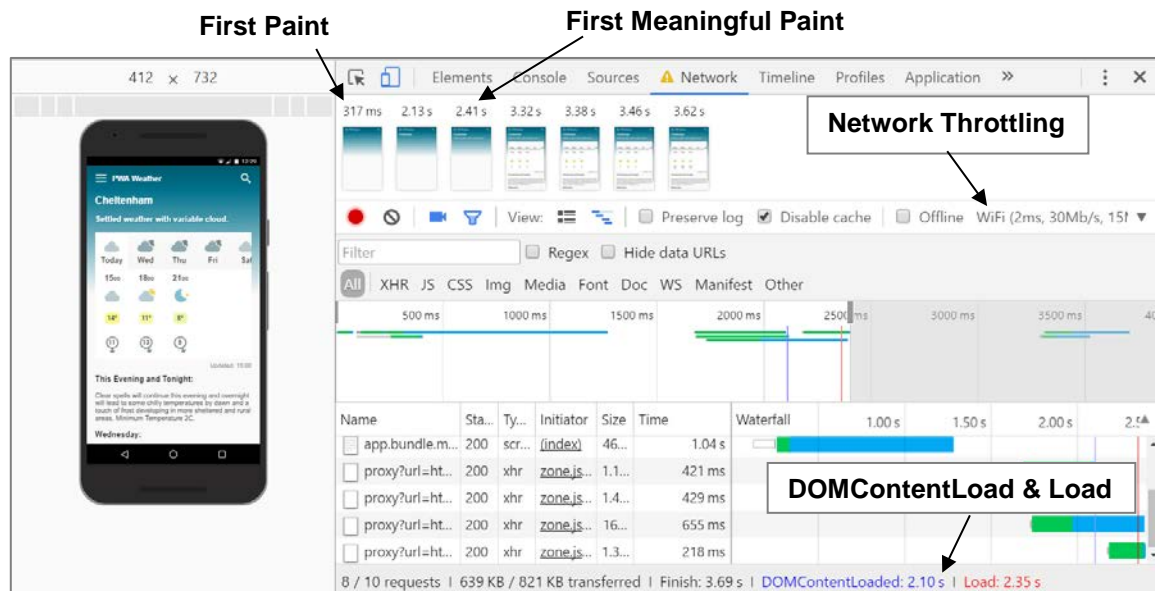


Figure 4 - Chrome Developer Tools: how load tests were measured

First Paint is represented by the first screenshot within the timeline, 317ms. First Meaningful Paint occurs at 2.41ms, the time when the forecast text, “Settled weather with variable cloud”, is first visible.

3.1.1 Non-blocking JavaScript

The develop branch of the application is built using non-blocking JavaScript, with script references placed at the bottom of the body tag. These scripts were shifted to the head of the HTML to test the effect of blocking JavaScript.

		Low End Device				High End Device			
		Test 1	Test 2	Test 3	Avg	Test 1	Test 2	Test 3	Avg
Wifi	DOMContentLoaded	2.32	2.05	3.17	2.51	1.02	1.02	0.98	1.01
	Load	2.39	2.09	3.23	2.57	1.03	1.03	1.17	1.08
	First Paint	0.523	0.483	1.23	0.75	0.23	0.257	0.215	0.23
	First Meaningful Paint	2.8	2.5	3.74	3.01	1.58	1.37	1.34	1.43
3G	DOMContentLoaded	4.61	4.63	4.48	4.57	3.38	3.33	3.36	3.36
	Load	4.72	4.73	4.53	4.66	3.39	3.34	3.37	3.37
	First Paint	0.786	0.885	0.735	0.80	0.275	0.247	0.26	0.26
	First Meaningful Paint	5.44	5.21	4.87	5.17	3.64	3.65	3.6	3.63

Table 2 - Test results for non-blocking JavaScript

		Low End Device				High End Device			
		Test 1	Test 2	Test 3	Avg	Test 1	Test 2	Test 3	Avg
Wifi	DOMContentLoaded	2.96	3.59	2.65	3.07	1.03	0.981	1	1.00
	Load	3.05	3.66	2.7	3.14	1.05	0.991	1.01	1.02
	First Paint	1.4	1.5	1.09	1.33	0.247	0.283	0.241	0.26
	First Meaningful Paint	3.97	4.67	3.11	3.92	1.25	1.3	1.22	1.26
3G	DOMContentLoaded	4.73	4.68	4.75	4.72	3.35	3.35	3.31	3.34
	Load	4.8	4.73	5.51	5.01	3.36	3.35	3.32	3.34
	First Paint	1.17	1.18	1.34	1.23	0.397	0.36	0.372	0.38
	First Meaningful Paint	5.98	5.09	4.84	5.30	3.58	3.7	3.59	3.62

Table 3 - Test results for blocking JavaScript

Tables 2 and 3 show the test results for non-blocking and blocking for the identified metrics. Each test was repeated 3 times, for each combination of network connection/device, and averaged. A graphical representation of averaged results is shown in figures 5 and 6.

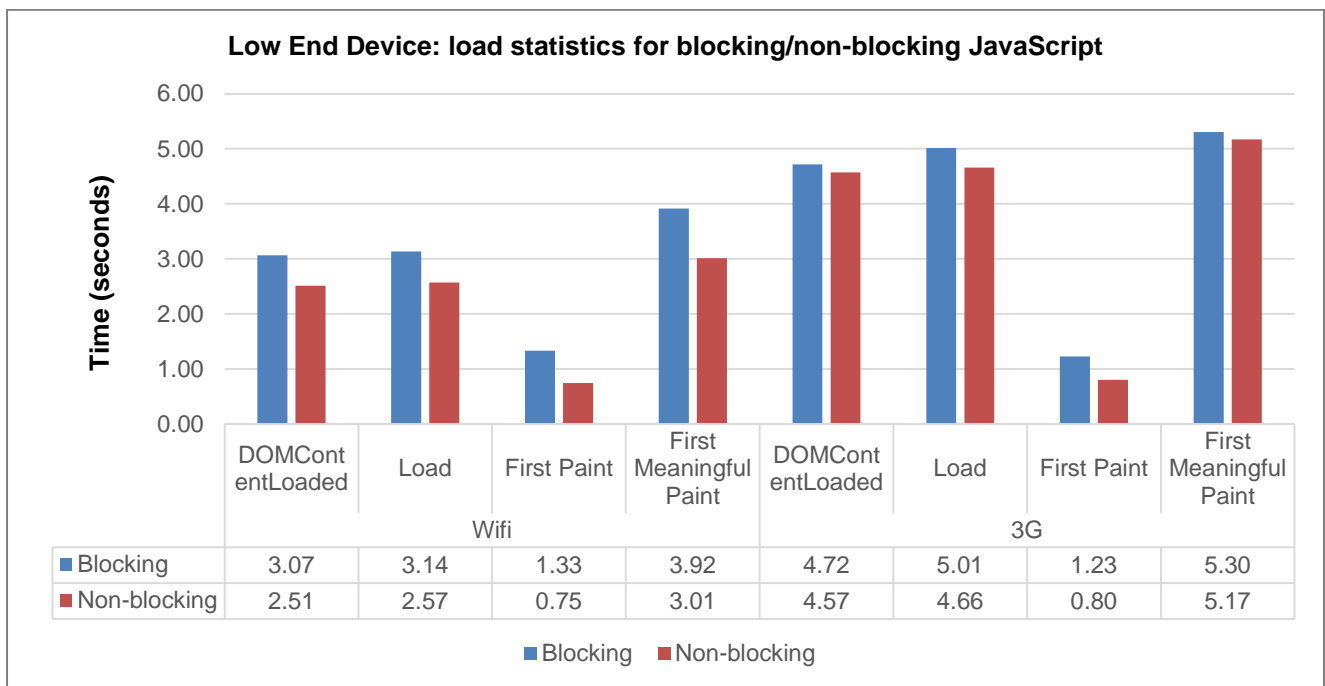


Figure 5 - Load statistics for blocking/non-blocking JS on a Low-End device.

Figure 5 shows non-blocking JavaScript to have a positive effect on initial load on a low-end device. Secondary research by Firtman (2016, p.289) within the literature review suggested the First Paint metric would be the most affected as the rendering of the initial UI would not be blocked. In the case of a low-end device this research opinion is supported, with the time taken for initial First Paint on WiFi down from 1.33s to 0.75s, a reduction of 0.58s (43.6%). On a 3G connection this is reduced from 1.23s to 0.8s (35%). Other metrics on a low-end device are positively affected, though these affects are less pronounced. The First Meaningful Paint metric on Wifi shows a 23% reduction from 3.92s to 3.01s; but shows the

largest time improvement of nearly a second. It is hypothesized that this is caused by the script responsible for the application shell blocking the initialisation of the AngularJS library and core logic of PWA Weather; slowing the display of meaningful content.

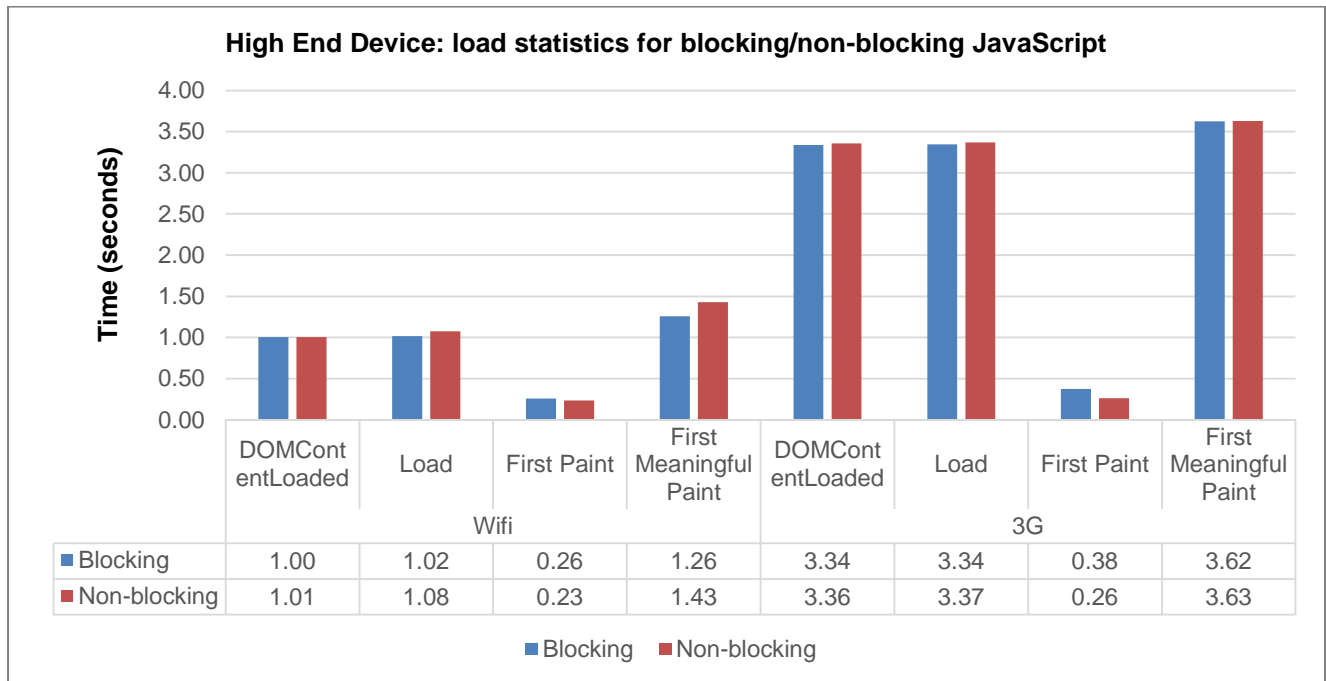


Figure 6 - Load statistics for blocking/non-blocking JS on a High-End device.

The high-end device, figure 6, does not show any clear improvement. Most metrics are close to identical, and in some cases non-blocking JavaScript was actually shown to increase load speed. First Paint does show a reduction on 3G from 0.38s to 0.26s (31.6%), but the time benefit is minimal. The faster processing speed of the high-end device appears to have reduced the positive effects making them negligible. For the performance considerations of low-end devices, which as noted by Housing.com (Google, 2016b) can be a large makeup of a PWA's user-base, a non-blocking structure should be considered best practice.

3.1.2 Minification

The develop branch contains a build task that minifies script files. This minification task was removed to test the performance difference. The file size changes are shown, table 4.

File	minified	non-minified
app-shell.min.js	5.6KB	10.5KB
app.bundle.min.js	466KB	1.3MB

Table 4 - File sizes for minified and non-minified code

The results show that the file size of the app.shell.min.js script was reduced from 10.5KB to 5.6KB following minification, a 46.7% decrease. The app.bundle.js showed greater percentage improvements, decreasing by 64.2%, down from 1.3MB to 466KB. These findings support those by Rajeev and Bakula (2015, p.672); who showed similar test results of minification more than halving the size of a JavaScript file. The effect of this reduction on load metrics is shown within table 5, and figures 7 and 8.

		Low End Device				High End Device			
		Test 1	Test 2	Test 3	Avg	Test 1	Test 2	Test 3	Avg
Wifi	DOMContentLoaded	4.37	3.59	4.38	4.11	1.45	1.36	1.71	1.51
	Load	4.47	3.67	4.49	4.21	1.46	1.37	1.73	1.52
	First Paint	2.37	1.14	1.6	1.70	0.24	0.25	0.758	0.42
	First Meaningful Paint	5.18	4.23	4.99	4.80	1.8	1.69	2.66	2.05
3G	DOMContentLoaded	9	9.63	11.17	9.93	7.81	7.86	7.86	7.84
	Load	9.1	9.71	11.24	10.02	7.82	7.87	7.87	7.85
	First Paint	0.709	1.01	0.997	0.91	0.238	0.307	0.317	0.29
	First Meaningful Paint	9.6	10.12	11.64	10.45	8.93	8.5	8.21	8.55

Table 5 - Test results for non-minified JavaScript files

Figure 7 shows minified code to have a positive effect on all load metrics for a low-end device. On a 3G connection First Meaningful Paint shows a reduction of 50.5% with minified code, changing from 10.45s to just over 5s. This is particularly significant given the secondary research in the literature review suggesting sites that load within 5 seconds can double the money made through advertising (DoubleClick, 2016, p.12). Without minification PWA Weather wouldn't meet this target, damaging profitability.

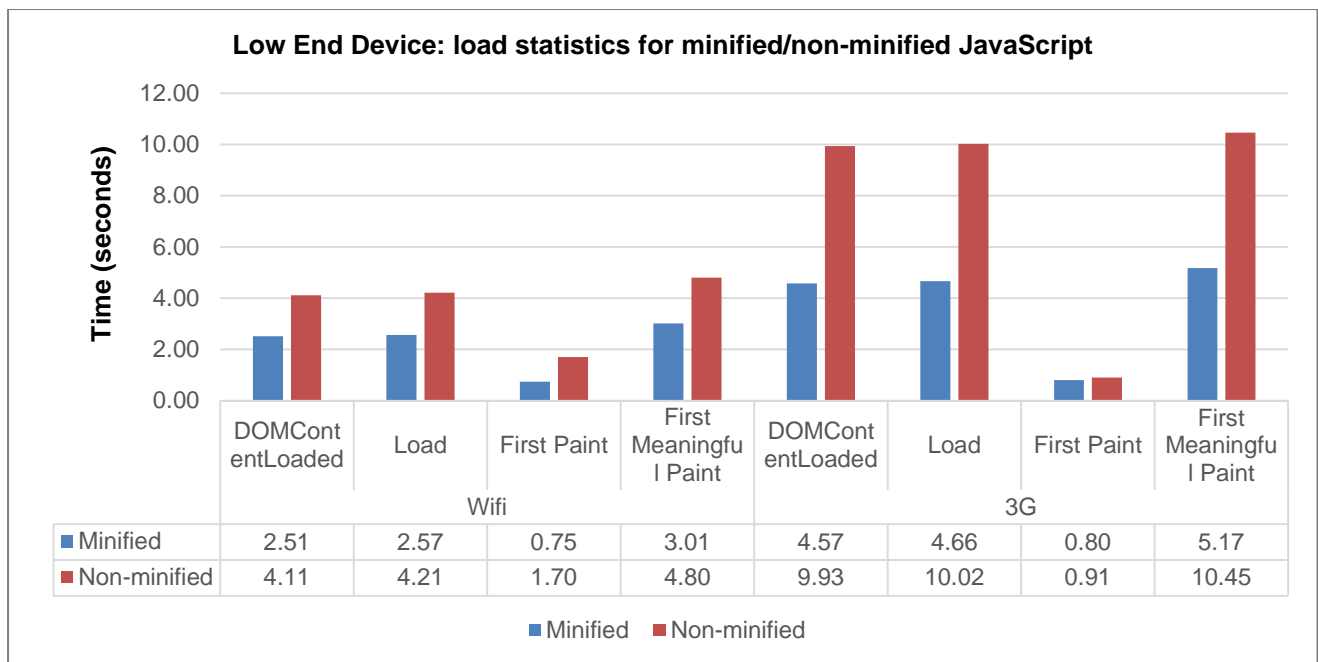


Figure 7 - Load statistics for minified/non-minified JS on a Low-End device.

Figure 8 shows similar improvements on a high-end device. In the case of First Meaningful Paint on 3G, the change is in a similar range, minification causing a reduction of 57.5%. It is noticeable however that despite the percentage changes being similar between the high-end and low-end device, the high-end has faster overall metrics; suggesting that the speed of the underlying hardware has an impact.

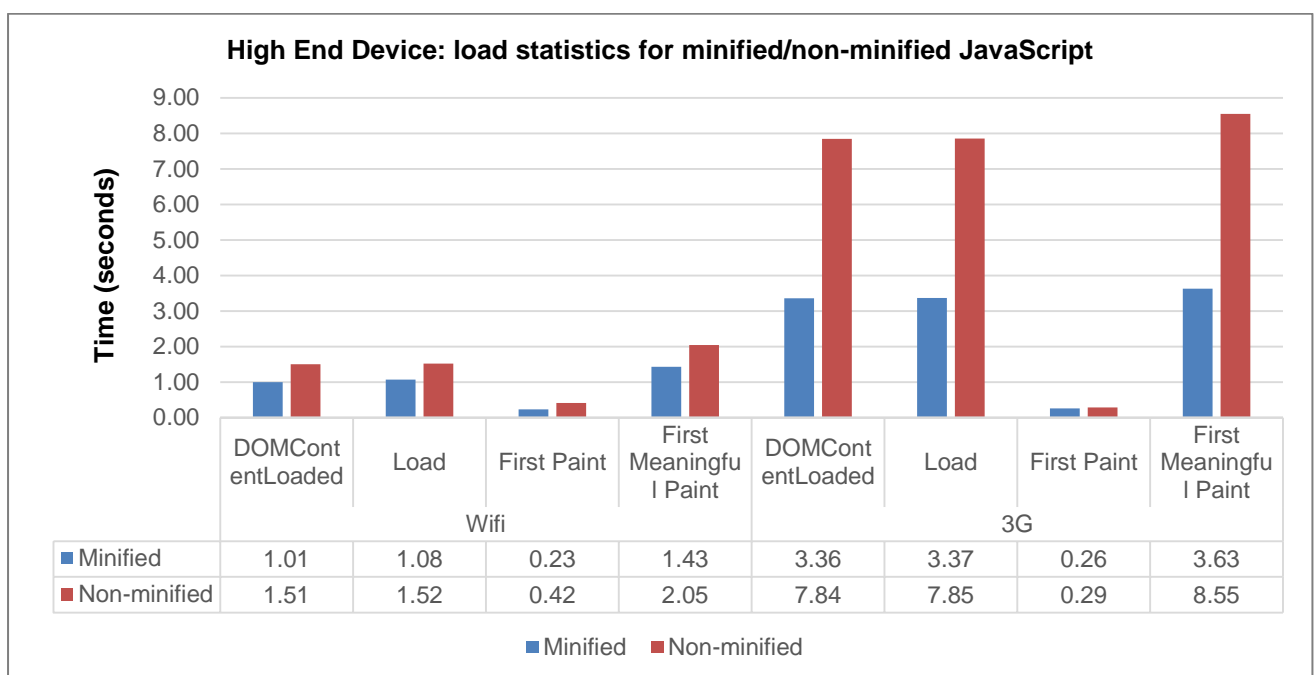


Figure 8 - Load statistics for minified/non-minified JS on a High-End device.

3.1.3 Tree-Shaking

The develop branch includes an AngularJS compilation step. Tree-shaking was added to this using the rollup.js library (Rollup, 2016). To test the effectiveness of tree-shaking on the reduction of file size, the following unused import was added to the main module, app.module.ts, for the Angular RouterModule and Routes classes:

```
import {RouterModule, Routes } from '@angular/router';
```

The compilation was run with and without tree-shaking. Each file was inspected and the tree-shaked version was shown to have all Angular Router source-code removed. The non-tree-shaked file contained all imported code. The resulting file sizes are shown in Table 6.

File	tree-shaked	non tree-shaked
app.bundle.min.js	466KB	532KB

Table 6 – File sizes for tree-shaked and non-tree-shaked code

A 12.4% reduction, 66KB, was shown to have been applied to the resulting bundle. File size reductions would be variable based on the proportion of unused imports or classes. Though some minor complications with configuration occurred, once the script is written it would be a minor process to include tree-shaking as part of any continuous integration. This would protect against redundant code from being deployed.

3.1.4 CSS: transform and will-change attributes

The develop branch uses the CSS transform and will-change optimisation (Google Chrome Developers, 2016) for the animation of the side navigation. Additional code was created with an animation using the CSS positional left property; Appendix 10 (section 9.10). The animation was run 3 times on each device and the framerate trace recorded in Chrome Developer Tools, see figure 9. 10 frames were taken from each trace, and averaged to provide an overall frames per second metric. The standard deviation was also measured and averaged to provide an indication of the framerate stability; a requirement of animation within RAIL (Kearny, 2016).

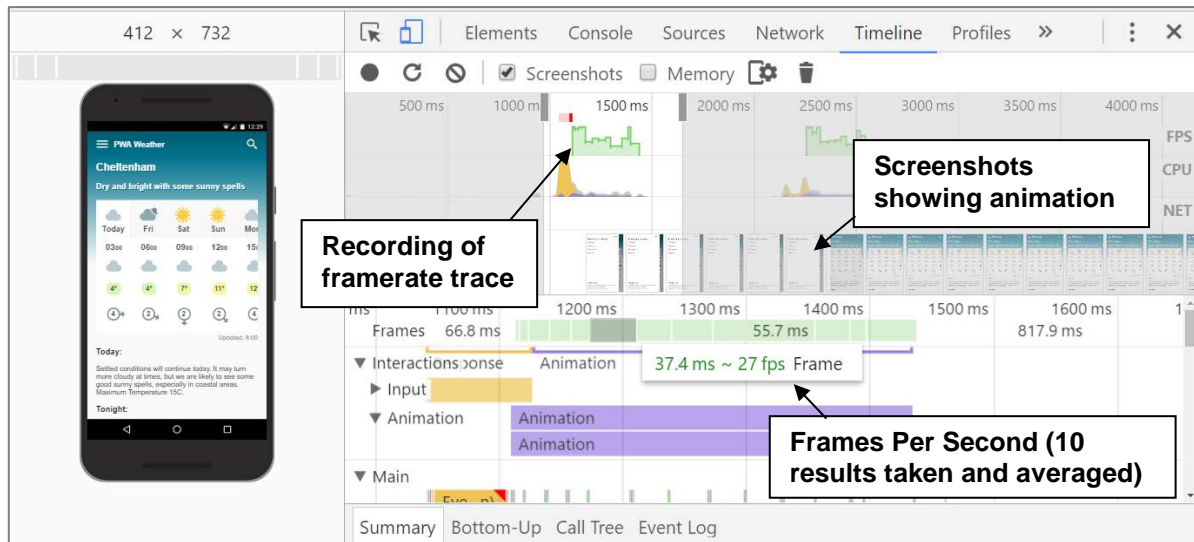


Figure 9 - Chrome Developer Tools: how framerate was measured

Figure 10 shows the averaged frames per second scores and standard deviation for both transform/will-change and left positioning. Tables 7 and 8 show the individual test runs:

Low End Device (using transform/will-change)												
Test Frames	1	2	3	4	5	6	7	8	9	10	Avg	Standard Deviation
Frames Per Second	33	92	52	29	64	73	60	59	60	26	54.80	20.68
	44	60	80	50	78	55	62	53	65	61	60.80	11.44
	31	49	29	65	78	58	59	51	73	61	55.40	16.04
High End Device (using transform/will-change)												
Test Frames	1	2	3	4	5	6	7	8	9	10	Avg	Standard Deviation
Frames Per Second	49	31	36	80	46	32	66	61	58	61	52.00	16.06
	46	31	61	61	55	64	59	59	66	59	56.10	10.37
	47	34	61	41	48	71	55	66	57	58	53.80	11.36

Table 7 - Test results for transform/will-change attributes

Low End Device (using positioning)												
Test Frames	1	2	3	4	5	6	7	8	9	10	Avg	Standard Deviation
Frames Per Second	14	10	72	30	30	63	57	60	59	59	45.40	22.23
	28	20	66	61	60	30	57	63	59	53	49.70	16.89
	22	16	64	26	57	37	59	26	34	31	37.20	16.88
High End Device (using positioning)												
Test Frames	1	2	3	4	5	6	7	8	9	10	Avg	Standard Deviation
Frames Per Second	23	65	59	56	59	67	53	29	67	53	53.10	15.25
	62	57	29	29	56	63	57	29	58	59	49.90	14.59
	58	62	55	61	62	29	58	53	61	28	52.70	13.10

Table 8 - Test results using CSS left positioning

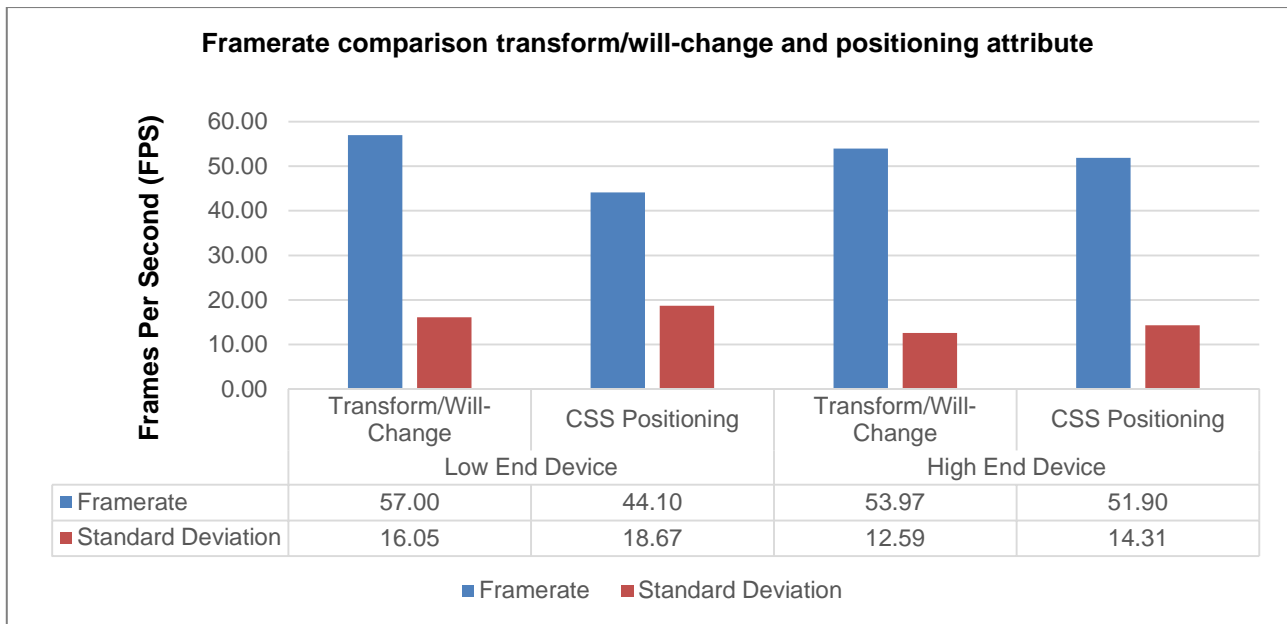


Figure 10 - Framerate statistics for transform/will-change and positioning CSS

Looking at figure 10 there is a clear improvement in overall framerate using the transform/will-change attributes on both the low-end and high-end device. The improvement is greater on the low-end device, an increase of 19.9 FPS, from 44.1 FPS to 57 FPS. This improvement was also visually noticeable when testing, the transform/will-change animation appearing more stable; backed by the low standard deviation scores. The difference is less clear in the high-end device, with only a small 2.07 FPS increase; suggesting that the processing power of the high-end device is able to better handle the redrawing of the page layout described in the literature review (Google Chrome Developers, 2016).

3.1.5 CSS: pointer-events and opacity attributes

The develop branch uses the CSS pointer-events and opacity optimisation (Google Chrome Developers, 2016) for the animation of the search results panel. Additional code was created with an animation using the CSS visibility attribute; Appendix 10 (section 9.10). Tables 9 and 10 show the individual framerate tests; figure 11 the comparison graph.

Low End Device (with pointer-events/opacity)												
Test Frames	1	2	3	4	5	6	7	8	9	10	Avg	Standard Deviation
Frames Per Second	58	59	59	54	61	61	58	69	60	57	59.60	3.89
	55	60	59	63	61	69	60	59	60	29	57.50	10.63
	24	55	65	58	55	93	60	62	59	61	59.20	16.54
High End Device (with pointer-events/opacity)												
Test Frames	1	2	3	4	5	6	7	8	9	10	Avg	Standard Deviation
Frames Per Second	37	30	59	61	29	48	77	60	59	62	52.20	15.71
	30	58	29	58	63	59	59	29	29	61	47.50	15.78
	61	57	59	58	60	31	53	77	59	56	57.10	11.19

Table 9 - Test results for pointer-events/opacity

Low End Device (with visibility)												
Test Frames	1	2	3	4	5	6	7	8	9	10	Avg	Standard Deviation
Frames Per Second	4	51	67	52	72	60	50	68	64	60	54.80	19.39
	7	48	53	73	70	55	62	55	50	83	55.60	20.44
	9	20	69	63	60	62	62	60	61	30	49.60	21.39
High End Device (with visibility)												
Test Frames	1	2	3	4	5	6	7	8	9	10	Avg	Standard Deviation
Frames Per Second	15	51	63	60	60	57	54	28	63	32	48.30	17.02
	31	59	18	61	59	58	53	56	60	59	51.40	14.68
	32	59	29	75	57	57	58	28	59	59	51.30	15.85

Table 10 - Test results for CSS visibility attribute

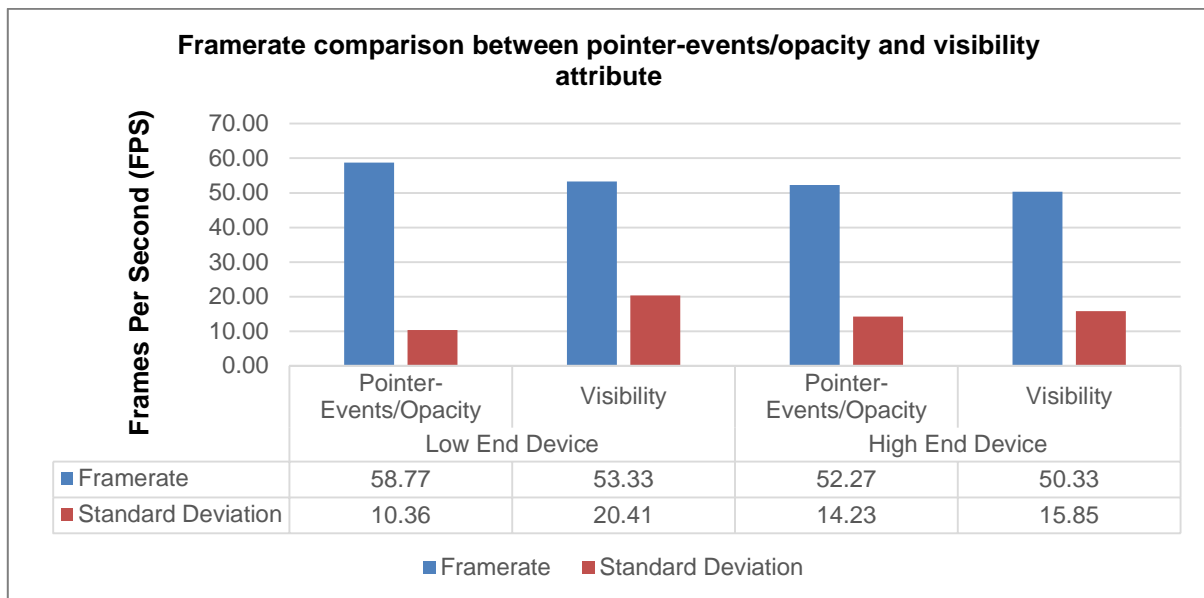


Figure 11 - Framerate statistics for pointer-events/opacity and visibility CSS

The overall framerate shows an increase similar to the transform/will-change test. There is an increase of 5.4FPS with pointer-events/opacity properties on the low-end device. The standard deviation is also half that of the CSS visibility attribute, suggesting the stability

also improved. With the high-end device, the test result changes are again minimal and no visual difference was identified between the two approaches.

3.1.6 Web Workers

The develop branch uses a filter to allow searching for a location from around 9000 pre-cached UK locations. This is achieved without a web worker. To test whether multi-threading with a web worker would improve framerate, code changes were made; Appendix 11 (section 9.11). Figure 12 shows the final comparison graph, tables 11 and 12 show the individual framerate tests.

Low End Device (with web worker)												
Test Frames	1	2	3	4	5	6	7	8	9	10	Avg	Standard Deviation
Frames Per Second	32	15	9	16	3	3	2	5	20	23	12.80	10.13
	26	10	8	1	1	4	20	3	4	4	8.10	8.45
	16	26	22	9	12	21	3	22	2	8	14.10	8.53
High End Device (with web worker)												
Test Frames	1	2	3	4	5	6	7	8	9	10	Avg	Standard Deviation
Frames Per Second	63	51	23	49	18	11	17	26	30	27	45.00	17.05
	61	42	62	96	66	56	61	55	25	92	61.60	20.89
	29	57	17	18	3	14	30	33	18	18	23.70	14.64

Table 11 - Test results for search typing with Web Worker

Low End Device (no web worker)												
Test Frames	1	2	3	4	5	6	7	8	9	10	Avg	Standard Deviation
Frames Per Second	16	29	10	4	18	3	5	7	7	25	12.40	9.17
	14	17	32	20	7	19	48	62	67	51	33.70	21.60
	26	26	34	11	15	15	5	7	10	21	17.00	9.45
High End Device (no web worker)												
Test Frames	1	2	3	4	5	6	7	8	9	10	Avg	Standard Deviation
Frames Per Second	50	42	41	47	23	29	55	50	76	59	67.43	15.01
	39	34	36	22	49	72	59	51	65	14	44.10	18.55
	56	26	50	31	82	59	33	59	58	57	51.10	16.83

Table 12 - Test results for search typing without Web Worker

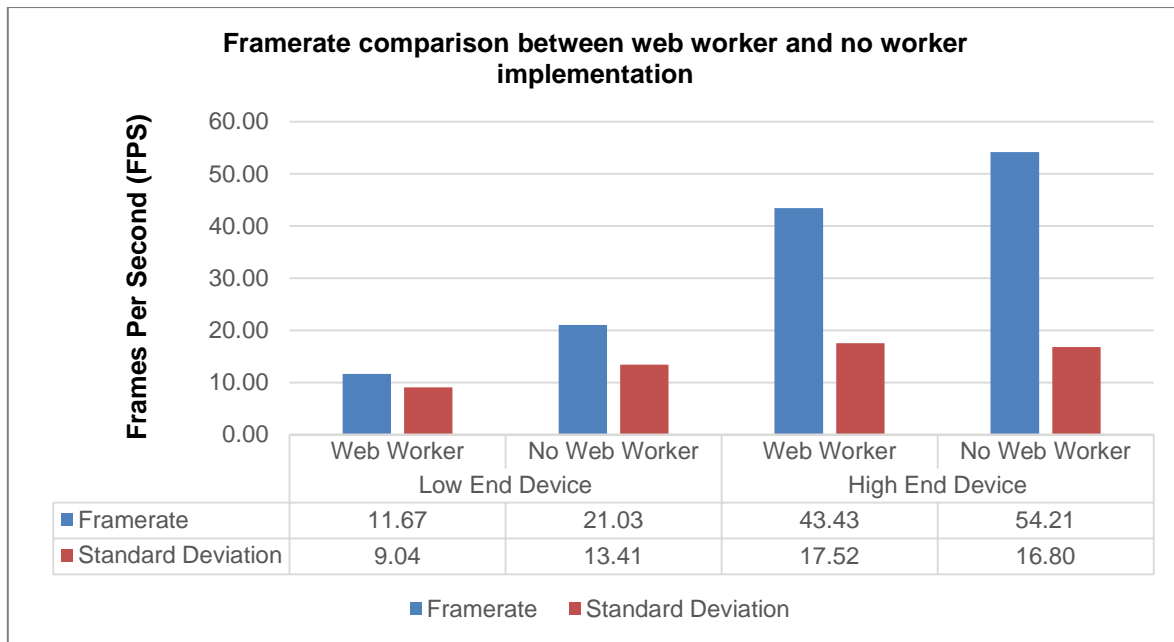


Figure 12 - Framerate statistics for web worker / no web worker

The results, figure 12, suggest that the web worker in this test case had a negative effect; reducing the framerate on both the low-end and high-end device. Whilst the stability is marginally better noted by the standard deviation, the frames per second rate almost halved, 44%, on the low-end device, and reduced by 20% on the high-end device. This conflicts with the results by Wenzel and Meinel (2015) who used web workers effectively for the parsing of data.

As the worker operates by message and not shared memory (Wenzel and Meinel, 2015, p.140), an explanation for this could be the additional start-up time and delay of sending the data to the worker thread. In the Wenzel and Meinel use case, the extra thread was executed as a one-time event. Within PWA Weather, threading was utilised as a user typed. This leads to multiple messages being sent. This appears to limit the effectiveness of the spawned thread; further research would be needed to establish performance gains in such a case.

3.1.7 GraphQL

An additional branch was created with the GraphQL implementation, Appendix 12 (section 9.12). The develop branch retrieves data for constructing of the forecast table and text summary from multiple calls to the REST API. A GraphQL service was implemented using a Node.js server. This facilitated combining multiple endpoints into a single request. The resulting GraphQL request and response is shown in table 13.

GraphQL Query <pre> { location(id: "310047") { id DailyForecasts { issueTime days { date weatherType } } ThreeHourlyForecasts { days { date timesteps { time weatherType temp windSpeed windGust windDirection } } } } } </pre>	Response <pre> { "data": { "location": [{ "id": "310047", "DailyForecasts": { "issueTime": "2017-04-03T13:00:00Z", "days": [{ "date": "2017-04-03T00:00:00.000Z", "weatherType": "7" } ...] }, "ThreeHourlyForecasts": { "days": [{ "date": "2017-04-03T00:00:00.000Z", "timesteps": [{ "time": "2017-04-03T09:00:00Z", "weatherType": "3", "temp": "11", "windSpeed": "9", "windGust": "13", "windDirection": "SSW" } ...] } ...] } }] } } </pre>
---	--

Table 13 - GraphQL Query and Response for data from the Met Office DataPoint API.

With a GraphQL request the client specifies the exact data it needs. For instance, nested within the DailyForecasts are ‘date’ and ‘weatherType’ options. The response then only contains this data. Other options, such as maximum temperature, can be requested client side with no changes to the server-side API.

Tables 14 and 15, figures 13 and 14, show the load test results for both GraphQL and the regular REST API calls. In addition to DOMContentLoaded and First Meaningful Paint, the ‘Time to First Byte’ metric (time the client spends waiting for a server response) and the total resource load time have been assessed.

		Low End Device				High End Device			
		Test 1	Test 2	Test 3	Avg	Test 1	Test 2	Test 3	Avg
Wifi	DOMContentLoaded	2.99	2.84	2.83	2.89	1.08	1.01	0.915	1.00
	First Meaningful Paint	4.02	3.97	3.63	3.87	1.46	1.38	1.31	1.38
	Time to First Byte	0.2082	0.4221	0.2327	0.29	0.2286	0.2282	0.2558	0.24
	Resource Load Time	0.49827	0.3527	0.387	0.41	0.2602	0.2563	0.322	0.28
3G	DOMContentLoaded	4.42	4.45	4.45	4.44	3.36	3.35	3.41	3.37
	First Meaningful Paint	5.19	5.28	5.1	5.19	3.85	3.76	3.59	3.73
	Time to First Byte	0.258	0.1985	0.1889	0.22	0.3216	0.2595	0.2316	0.27
	Resource Load Time	0.32697	0.2982	0.2935	0.31	0.4081	0.3233	0.3245	0.35

Table 14 - Test results using the GraphQL service.

		Low End Device				High End Device			
		Test 1	Test 2	Test 3	Avg	Test 1	Test 2	Test 3	Avg
Wifi	DOMContentLoaded	3.22	2.91	2.9	3.01	1.17	0.983	0.914	1.02
	First Meaningful Paint	3.69	3.82	3.51	3.67	1.64	1.29	1.28	1.40
	Time to First Byte	0.43745	0.3017	0.3065	0.35	0.6	0.5361	0.6208	0.59
	Resource Load Time	0.57556	0.3656	0.4344	0.46	0.6924	1.2909	0.813	0.93
3G	DOMContentLoaded	5.1	5.04	4.97	5.04	3.4	3.36	3.4	3.39
	First Meaningful Paint	5.56	5.63	5.5	5.56	3.79	3.72	3.63	3.71
	Time to First Byte	0.37174	0.3697	0.3341	0.36	0.4354	0.4719	0.4236	0.44
	Resource Load Time	0.54864	0.5459	0.4585	0.52	0.5039	0.548	0.4805	0.51

Table 15 - Test results using the REST API

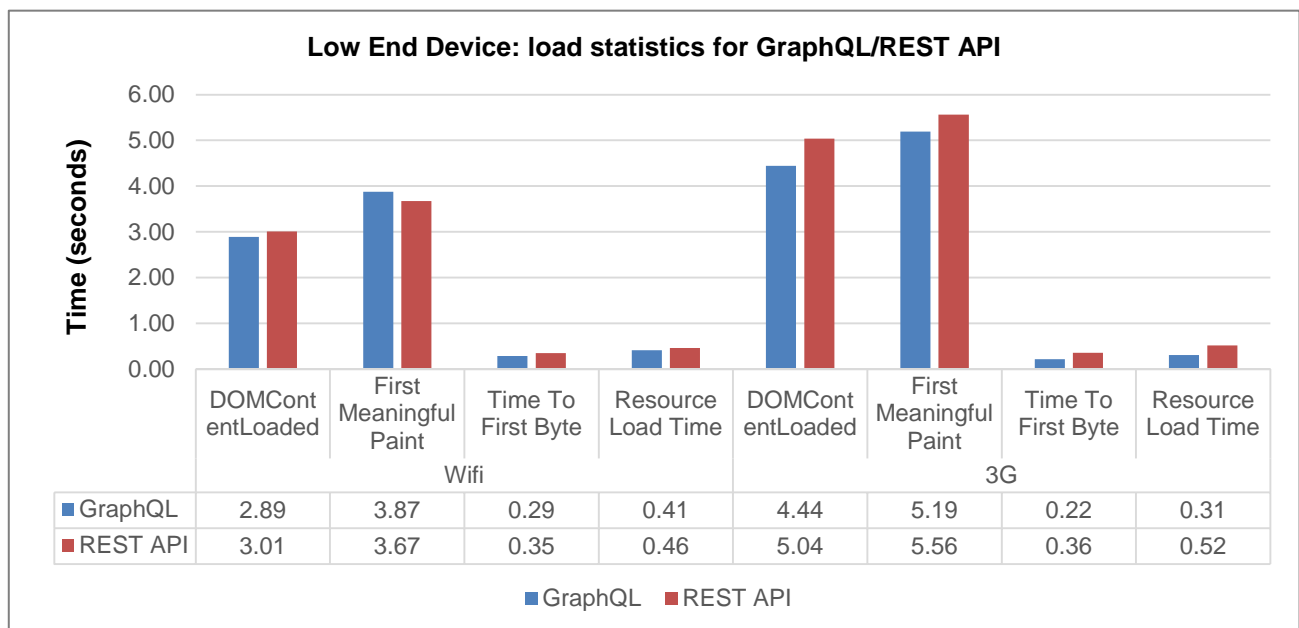


Figure 13 - Load statistics for GraphQL/REST API on a Low-End device.

GraphQL has shown to offer a small performance gain where making multiple REST API calls is necessary; confirming the research noted by Cederlund (2016, p.82) in the literature review. In response to the limitation of that research, this paper has tested the results on a 3G connection. On 3G the resource loading time decreased by 40% for the low-end device,

from 0.52s to 0.31. This also positively affected the DOMContentLoaded and First Meaningful Paint metrics. On the low-end device changes were minimal for Wifi, but the resource loading time decreased from just under a second, 0.93s, to 0.28s for the high-end device under the same conditions. The graphql query avoids over-fetching data. This reduces the file size and makes for a quicker request. For larger APIs where the ability to change them is difficult or impossible, GraphQL would make a performant and scalable solution.

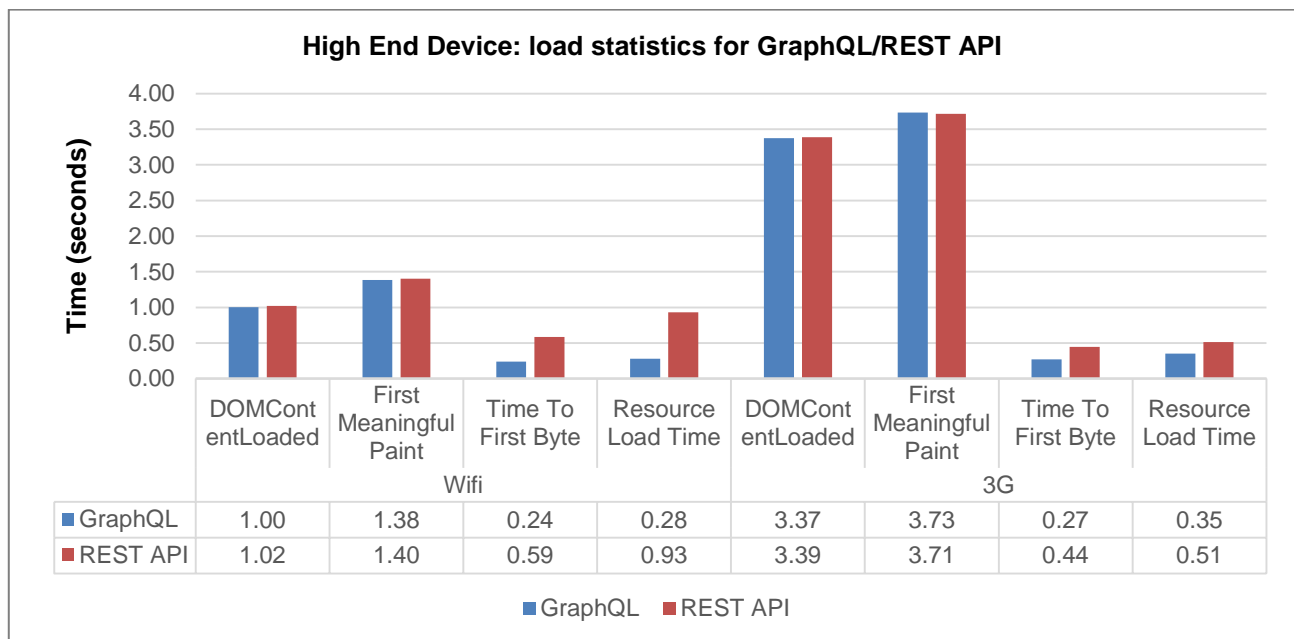


Figure 14 - Load statistics for GraphQL/REST API on a High-End device.

4. Questionnaire

4.1 Design

A questionnaire was designed to gain end user feedback on the performance of PWA Weather; drawing comparisons with native applications. Given that some individuals have no understanding of native and web based applications, the decision was taken to allow them to draw comparisons to any existing weather application used on their smartphone. This would ensure no requirement for users to install additional unwanted applications and increase the likelihood of participation.

This decision was also taken to facilitate comparisons to a broader set of competitor applications. Comparisons to just a single native application may have resulted in bias if that application itself has notable performance issues. A full copy of the final design is included in Appendix 5 (Section 12.5). Three questions were included to gauge opinion on metrics that form part of the RAIL methodology; the response, animation and load life cycle events (Kearny, 2016). Rationale for each is detailed in table 16.

Question	Rationale
Find and select the PWA Weather icon on your home screen. Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather loads?	This question was included to gauge opinion on how quickly PWA weather was able to load compared to existing native applications, linking to the load metric of the RAIL methodology (Kearny, 2016). It also allowed users to test the 'add to homescreen' process of a PWA and would identify any positives or difficulties associated.
Use PWA Weather to search for a forecast. Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather returns a forecast?	The intent of this question was to identify any performance issues associated with input delay, the 'response' metric of the RAIL methodology (Kearny, 2016), and also any issues with external data retrieval from the API.
Click the burger menu in the top left corner and select any one of the previous locations. Compared to your 'frequently used weather application', how would you rate the performance of animations within PWA Weather?	Selecting the burger menu animates the side navigation as a slide-in from the left. At this point the user should have also seen the search panel and burger menu animate when searching for a forecast. The intent of this question therefore was to gauge any opinions on the animation metric of the RAIL methodology (Kearny, 2016), establishing whether animations within the web view have any discernible lag to regular end users.

Table 16 – Rationale for main performance questions of the end user questionnaire.

4.2 Results and Discussion

A full copy of all questionnaire responses is provided in Appendix 6 (Section 12.6). The questionnaire was made available online (eSurvey, 2017). 37 individuals, including Met Office employees and University of Gloucestershire students, participated. Of the 37, 31 completed the questionnaire fully. 6 individuals completed the introductory questions, but did not attempt the testing of PWA Weather. No reasoning was provided. It could be hypothesized that this is the result of individuals not understanding the 'add to homescreen' process. Alternatively, it could be a lack of time, interest, or a limitation of the questionnaire design not providing enough information. 31 individuals completed all performance questions; 20 additionally provided answers to an open question providing further insights.

4.2.1 Quantitative Data

The quantitative data has been organised and discussed below:

Q.2 Please name the weather application you use most frequently

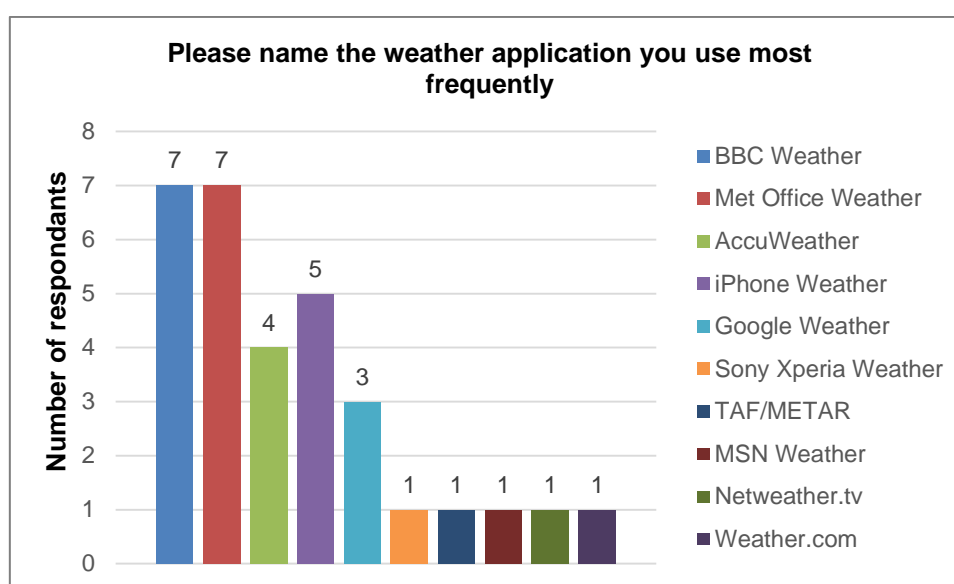


Figure 15 - Frequently used weather applications of the questionnaire participants.

Figure 15 shows 45% of users collectively compared against the BBC Weather and Met Office Weather. A further 38% compared against AccuWeather, iPhone Weather and Google Weather collectively. These applications share similar feature sets to PWA Weather and help to ensure the validity of later performance comparisons.

Q.4 How would you rate the level of your digital skills (your ability to use computers, smartphones, apps, the internet etc..)?

Figure 16 shows the digital skill level of the participants. No individual rated as a novice or advanced beginner. This is likely a limitation of the questionnaire distribution, though could indicate the skill level required to add the PWA to the homescreen was a barrier preventing the less experienced from participating.

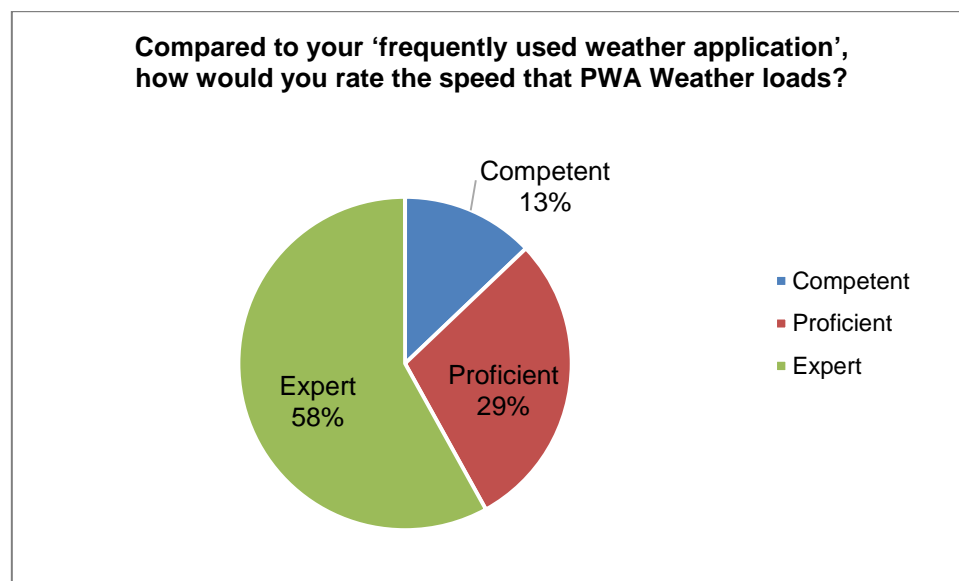


Figure 16 - Digital skill level of the questionnaire participants.

Of the participants that completed, 58% rated themselves as an expert level, indicating they have professional experience with technology. These users, especially those with software development expertise, may provide a more technical insight with their responses, but may also bias the results. They would be more aware of development practices, and potentially either be more forgiving or critical of web application performance. This may not reflect the opinion of a standard application user. 13% rated themselves as competent and 29% as proficient, making a combined 42% of participants listed as non-expert users. Results from the expert and non-expert groups have been compared in subsequent questions to ascertain any such bias.

Q.5 Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather loads?

Figure 17 shows the ratings grouped by expert, non-expert and all users for the load speed of PWA Weather compared to existing applications.

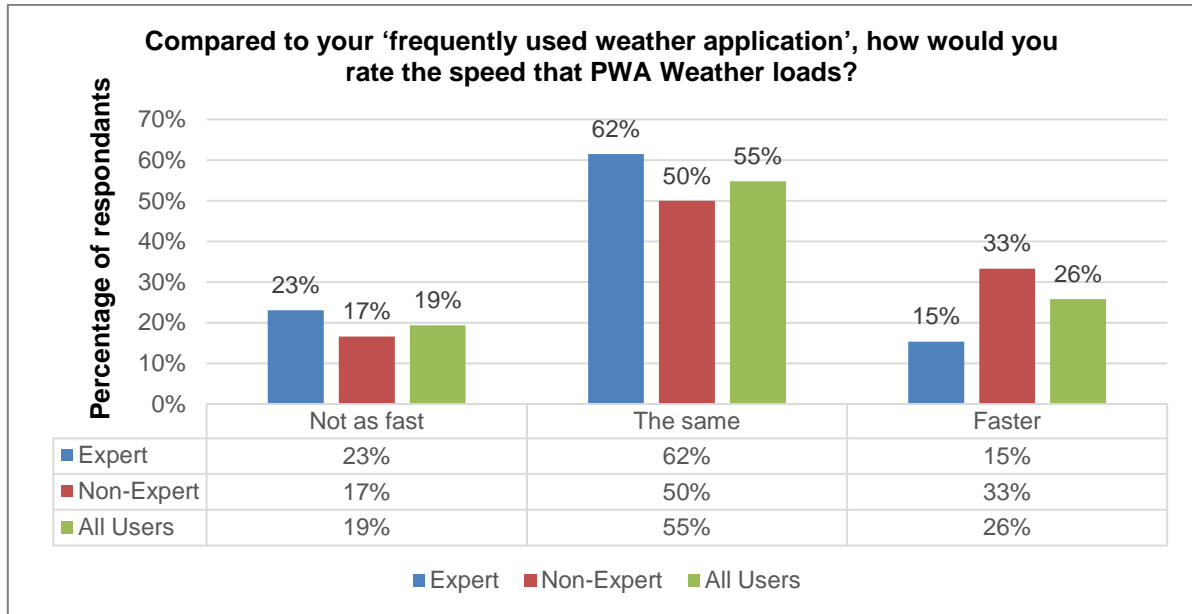


Figure 17 - User responses comparing the load speed of PWA Weather to a frequently used weather application.

The spread of responses for this question was fairly balanced amongst expert and non-expert users, with around 81% of all users rating the load speed as either comparable or faster than their frequently used application. Fewer expert users rated the experience as faster, 15% compared to 33%; potentially more critical of the performance given their experience.

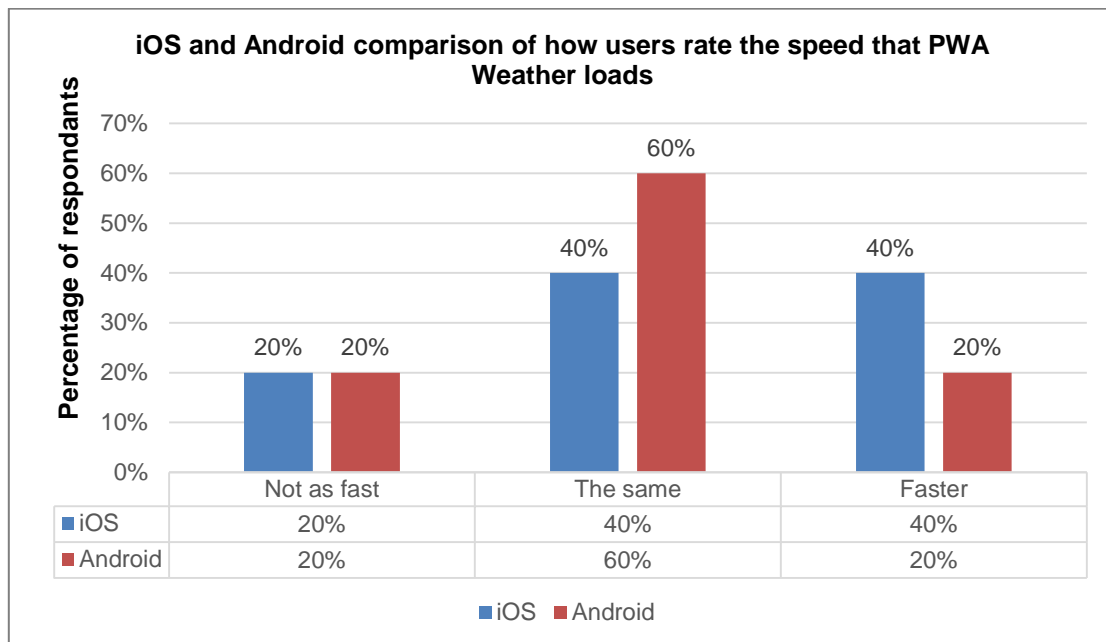


Figure 18 - Responses from all users comparing the load speed of PWA Weather grouped by Android and iOS.

Figure 18 shows the same response split by iOS and android users. An equal share, 80%, rated the load performance as either the same or faster. Interestingly, a larger proportion of iOS users, 40%, compared to 20% of Android, rated the experience as faster. This is unexpected given Android has service worker support (Malovolta, 2016, p.2) and caching should improve load. An explanation may be the processing power of the average iOS device is faster. In the case of the optimisations earlier in this paper, hardware performance was shown to have a positive effect with the load times on the high-end phone; often half that of the low-end device.

Q.6 Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather returns a forecast?

Figure 19 shows the ratings for the speed that PWA Weather returns a forecast compared to existing applications. Again, the response was positive, with only 3% of all users rating the performance not as fast. There was a split between expert and non-expert users; experts were more likely to rate the speed faster, at 67%, whilst non-expert users rated the performance as the same, at 62%.

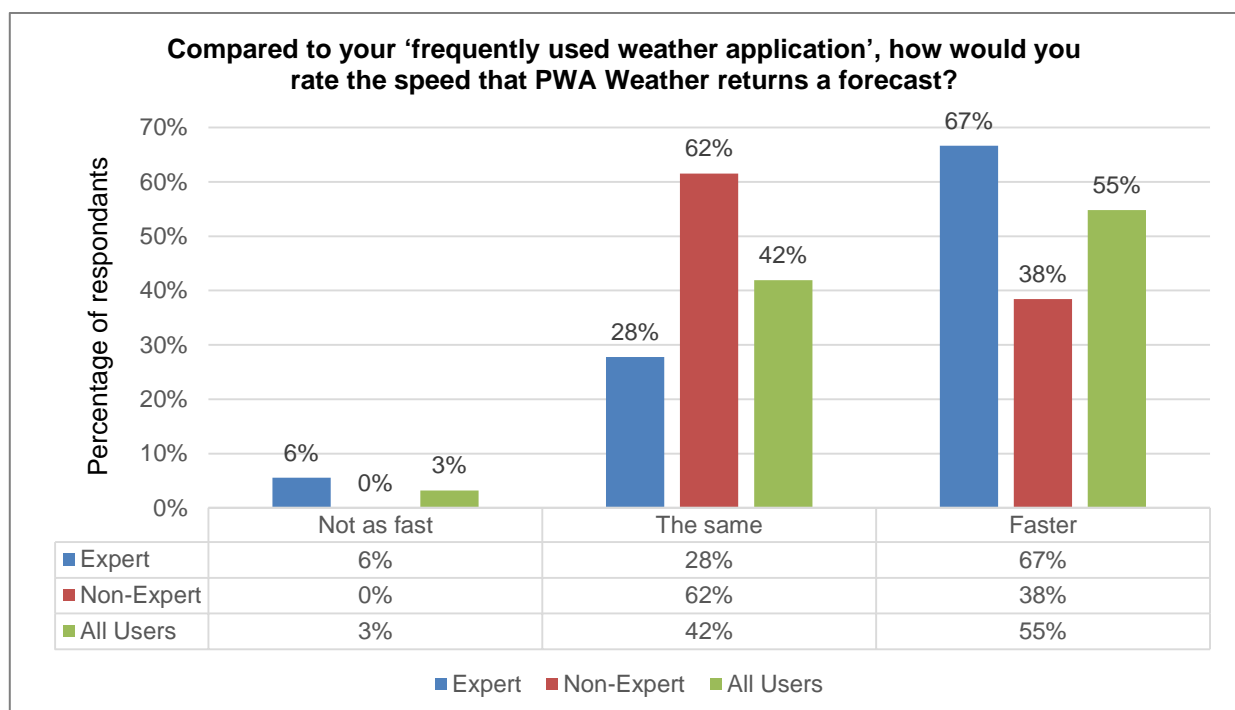


Figure 19 - User responses comparing the forecast load speed of PWA Weather to a frequently used weather application.

The split by iOS and Android, figure 20, doesn't reveal any significant trend, with only a marginal 10% more of Android users rating the experience as faster. Compared to the previous load test results this suggests iOS hardware may offer faster loading of the initial view, but once the application has loaded performance remains even.

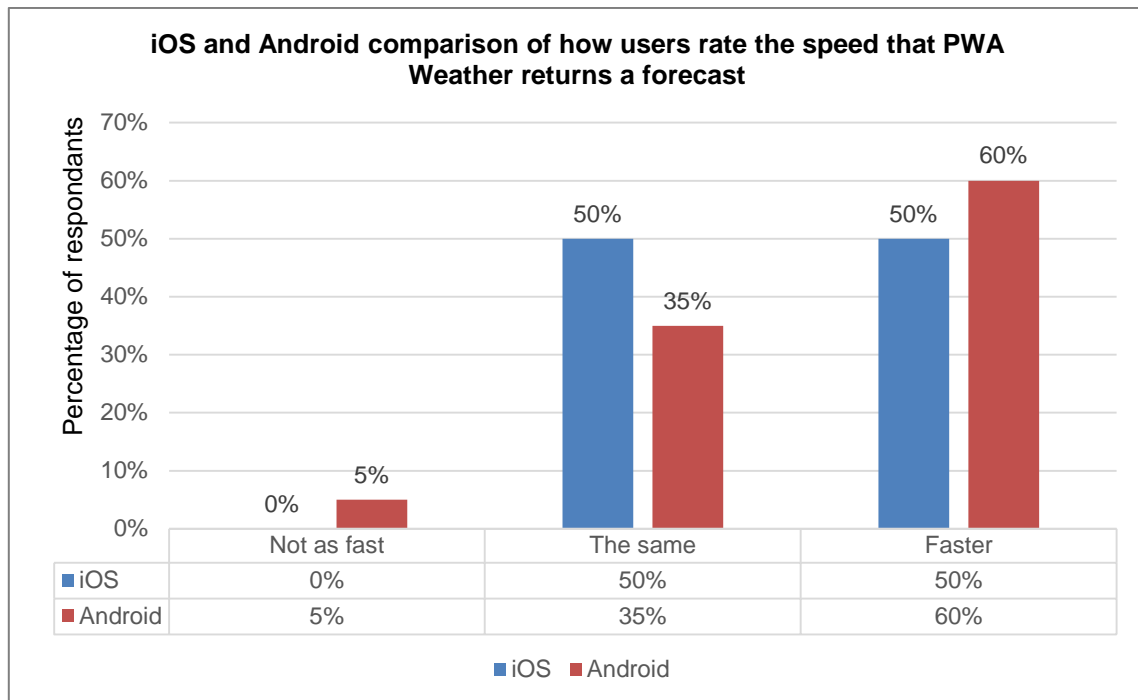


Figure 20 - Responses from all users comparing the forecast speed of PWA Weather grouped by Android and iOS.

Q.7 Compared to your 'frequently used weather application', how would you rate the performance of animations within PWA Weather?

Figure 21 shows ratings for the performance of animations within PWA Weather compared to existing applications.

The results are positive, with 88% of all users stating the animations within PWA Weather were either the same or smoother. Interestingly, 54% of non-expert users rated the experience as the same, compared to 56% of expert users rating the experience as smoother. This may indicate a limitation of the questionnaire design, given that one non-expert user also answered that they didn't know how to answer. It is possible that non-expert users were uncertain and therefore rated the experience as the same rather than picking an extreme.

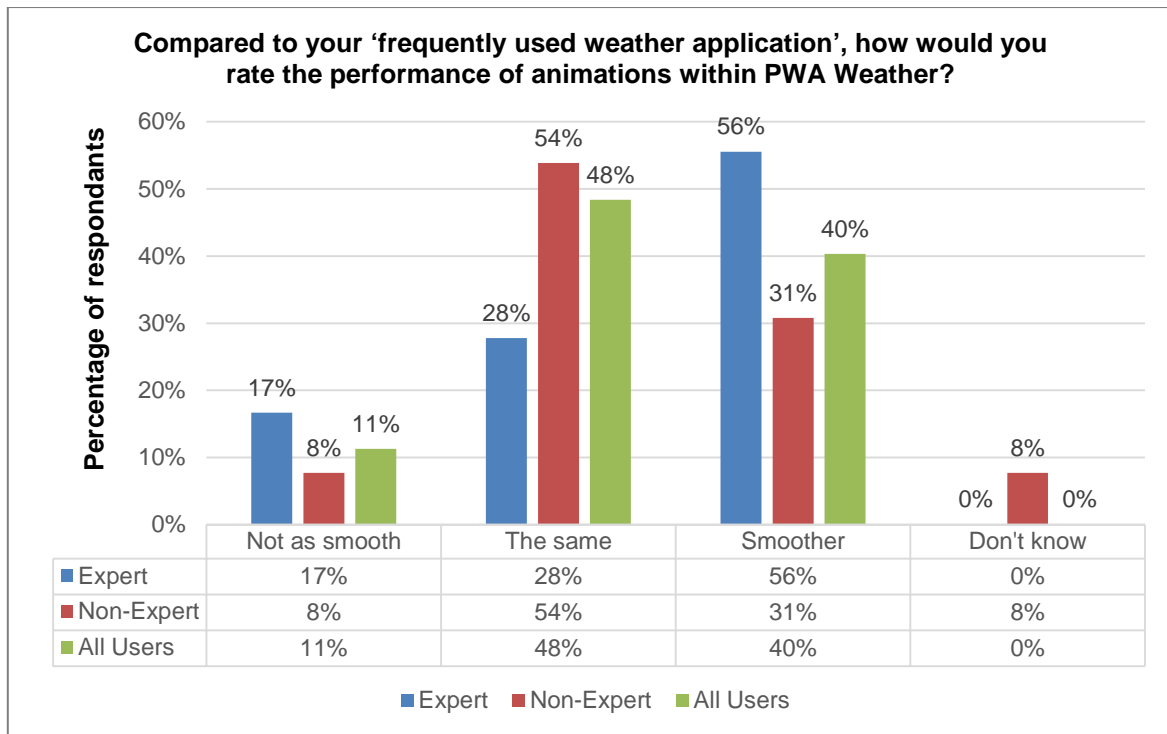


Figure 21 - User responses comparing the animation performance of PWA Weather to a frequently used weather application.

The comparison between iOS and Android, figure 22, shows a trend with Android users more willing to rate the animations as smoother at 55%. 50% of iOS users on the other hand rate the performance as the same. This potentially suggests the difference between CSS animations and native transitions is less distinct on Android devices.

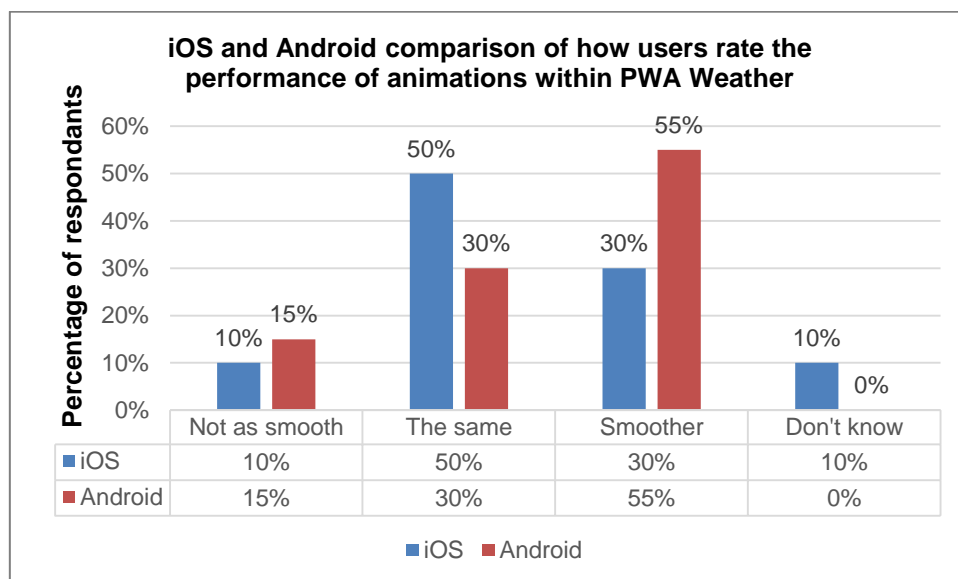


Figure 22 - Responses from all users comparing the performance of animations within PWA Weather grouped by Android and iOS.

4.2.2 Qualitative Data

Table 17 shows all responses provided in the final qualitative answer section of the questionnaire (Q.7 If you have any other comments about the performance of PWA Weather, please provide them below).

Seemed just as fast and easy to use as the BBC Weather application. The process of adding the app to my home screen was faster than if I had installed the application from Google Play store.
A fantastically fast and beautiful app. Well done :)
The load times of the forecast data & speed of search results is really impressive
The search functionality was slow to replicate typing, this could be an issue with my keyboard or the application
<ul style="list-style-type: none"> - Nice little weather application with simple, clear design; - Good concept to compare PWA vs Native app; - Speed/performance is comparable to Native apps, which is nice to see; - 'Install-able' feel of PWA is a great improvement over standard web app
What's there is great: For a limited-time development it's sufficiently 'rich' to be a good showcase and keep my interest for future possibilities. The sideways scrolling of weather symbols etc is great: symbols cut in half on the edge of the window make it really clear that the user should scroll.
I have experienced no problems with the performance.
I like having the extra descriptive information below the quick reference icons. It relays more information without being cluttered. Would recommend
A bit slow for the initial load, but very quick performance once loaded.
Couldn't actually use the search, nothing happened.
I prefer the layout of the pwa ; But I like being able to see an hour by hour weather report like the apple one has
Regarding animations; can't actually see any animation at all unless it's the very slight flicker when you view a new forecast. Too fast to really notice.
I like Applications that are simple but effective, and this one does just that. Plus, I like the convenience of having it available on my phone without having to install it.
Great product
Some features are laggy, can definitely tell it's a web-app. Header bar doesn't stick to the top all of the time. Parts of the application will jump rather than smoothly scroll. Some parts don't seem as responsive to touch compared to native applications.
I believe the performance is on par, if not better than the current applications available. I like the simplicity to give you the details fast and efficiently. Thus, creating a smooth experience for the end user.
Performance is outstanding. Intrigued to find that it works without a connection. Presume the locations are precached and so is the weather?; Still very fast, light and basic.
Met office has weather each hour where as this is every three. ; I prefer to scroll up and down rather than left to right to see how the days weather progresses.
Very good app, shows the relevant weather accurately. Would definitely use the finished app!
Looks good. I would like more features in future.

Table 17 - User responses to question "If you have any other comments about the performance of PWA Weather, please provide them below".

Table 18 below contains a discussion of the qualitative questionnaire data collected.

Researcher's Discussion of Positive Feedback
<p>User feedback on performance is generally found to be positive, with load times of the forecast and the efficiency of the app to provide the information quickly the main feedback provided. One expert user compared performance directly to native applications which suggests the performance to some is comparable. One expert user was also impressed with the ability that the application could function offline. It is likely that non-expert users were unaware of this fact, something that may need consideration when marketing PWAs in industry.</p> <p>3 responses were targeted at the 'add-to-homescreen' element of a PWA. Two users in particular indicating that the process was faster and more convenient than a native install. One expert user noted that the "'Install-able' feel of PWA is a great improvement over standard web app". However further research is required to find if non-expert users are happy with the process of adding the application.</p> <p>Comments on the usability of the PWA were also generally positive, with a significant focus on the simplicity and ease-of-use of the UI:</p> <ul style="list-style-type: none"> ▪ "simple, clear design", ▪ "simple but effective", ▪ "I like the simplicity to give you the details fast and efficiently", ▪ "I like having descriptive information below the quick reference icons. It relays more information without being cluttered", ▪ "fast and beautiful app". <p>These findings suggest that a PWA is able to compete with native usability.</p>
Researcher's Discussion of Negative Feedback
<p>Some negative feedback was provided in regards to performance issues. One user commented that the "search functionality was slow to replicate typing, this could be an issue with my keyboard or the application". Looking at the full response shown for this user, it is identified that they were using a Sony Xperia SP. This is a mid-range Android device and depending on the exact version could be several years old. However,</p>

performance problems are still surprising. The phone has a better specification than the low-end device used with the optimisation tests. This may be an issue with this particular device or a general issue with performance on lower-model smartphones that needs investigating.

Another user commented that the app was “A bit slow for the initial load”. The device used was a Samsung Galaxy S3, an older phone but not one with limited specs. This could be the age of the device or an issue with PWA Weather.

One response was negative on a few areas: “Some features are laggy, can definitely tell it's a web-app. Header bar doesn't stick to the top all of the time. Parts of the application will jump rather than smoothly scroll. Some parts don't seem as responsive to touch compared to native applications”. This was using an iPhone 6S. Limited testing has been performed on iOS devices and it is likely that there are some issues with PWA Weather. Other iOS users in the questionnaire have not commented on these however, so it may be a more subjective opinion on the user's part or a problem with this particular model of device that would need further investigation.

Summary

There is a general positive indication that the performance is suitable and in cases has matched that of native applications. Users have found the usability of the PWA to be on par with native apps. Whilst there have been some minor performance issues noted, these are not the norm and would need further investigation.

Table 18 - Discussion of qualitative questionnaire responses

5. Seminar and Interview Discussion

An unstructured interview with a group of software professionals was undertaken following the presentation of PWA Weather at a Met Office seminar on the 21st March 2017. An unstructured approach was chosen to allow views on complexity, scalability and viability of PWAs to be developed upon and revised. A group setting allowed for collaboration of ideas and for opinions to be peer reviewed.

11 individuals attended the meeting and of those 5 main individuals contributed in the discussion. There was a broad mix of job roles covered; a solution architect (responsible for the technical direction at the Met Office), two senior developers, a senior tester, and a product manager. This collected opinions from a business, technical and test/security perspective. As a means to start the interview and offer some direction, a set of 6 questions were devised:

- I. How would you consider the scalability of PWA Weather? Would it need any extra functionality that might cause performance problems?
- II. Do you know of any additional or better optimisations that should be considered?
- III. Are JavaScript frameworks (e.g. Angular) or CSS frameworks required or can it be done without?
- IV. Any on-device hardware/software features not supported by the web?
- V. Would marketing the PWA be viable? Discoverability by users an issue?
- VI. How important is security?

Questions in regards to the use of JavaScript frameworks, feature support and security were chosen to identify any potential scalability problems. A question on marketability was also asked to draw conclusions as to whether a PWA was a viable business prospect. A full transcript of the seminar presentation and interview is provided in Appendix 7 (Section 12.7). Slides demonstrated at the seminar can be viewed in Appendix 8 (Section 12.8). The main points raised by the participants within the interview have been discussed in table 18.

Interview Point Raised	Researcher's Analysis and Discussion
------------------------	--------------------------------------

Interview Point Raised	Researcher's Analysis and Discussion
<p>Lack of multi-threading and processing power available to a web-based application such as a PWA. Met Office native application involves the processing of images together for displaying rain radar layers on top of a map. Participant 1, a senior developer, found in their own investigation in a web context that processing such information “on-the-fly... would really kill the application” but commented that it is possible within a native app as there is “more access to the lower level operating systems”.</p> <p>When questioned by the researcher over the main cause of this being a lack of multi-threading within JavaScript, participant 1 agreed, stating that “you’re multi-threaded when you’re doing native apps so you can do more stuff simultaneously, but you can’t do that on the web”.</p>	<p>This opinion suggests agreement with the point made by Oh and Moon (2015, p.179), that the constructs of the JavaScript language can limit its efficiency compared to native compiled languages.</p> <p>Research by Wenzel and Meinel (2015, p.140) in the literature review of this paper suggested that the single-threaded nature of JavaScript was preventing the power of modern smartphones from being taken advantage of. The views of participant 1 show agreement with this research and that from a practical experience in industry the single-threaded construct is having a limiting effect.</p>
<p>“move some of that [processing] to the backend and do it, or use a web worker” (Participant 1). Participant 2, solutions architect, agreed that this might be plausible, but would need investigation. Participant 2 clarified that the native application is also currently processing images and returning colour values for each pixel to allow mapping to a key.</p> <p>In reference to whether that processing was possible in a PWA, Participant 2 was uncertain as to “whether you could do that efficiently with a JavaScript engine on a browser”, suggesting “I think that will improve as the hardware improves” but “that’s not going to be as performant as a native app would do”.</p>	<p>Notably from a practical perspective implementing web workers as part of the optimisation section of this paper (Section 3.1.6), performance improvements are not always guaranteed.</p> <p>Image manipulation, particularly in the case of displaying on a map, is likely to involve DOM access, which is not possible with web workers (Wenzel and Meinel, 2015, pp.141-142). As such it is unlikely that a PWA would be able to compete with a native application in such a case without significant time spent on further research.</p>

Interview Point Raised	Researcher's Analysis and Discussion
<p>But NO actual business need for intensive processing most of the time. Participant 2 stated that “most of the time I don’t, certainly for a weather style app, I don’t see any reason why we wouldn’t have been doing this [building PWAs]. I wanted to do this the first time around, but I wasn’t in charge of the app”. From a performance perspective, they stated that most applications are “just data volume, data sort of retrieval. It’s not overly complex in terms of sort of running big algorithms, across it”. Participant 2 suggested that from a development best practice view that if there was a need for running complex algorithms that these should be moved server-side.</p>	<p>The support for building PWA’s in most cases is promising. Whilst some apps are heavily reliant on video, image manipulation or other intensive processes that wouldn’t be achievable as of now, the fact that most applications are not reliant on such features opens the possibility for more PWA development. Moving algorithms server-side is a good best practice suggestion.</p>
<p>Web good for visually impaired: Participant 1 has a visual impairment, and noted that in PWA Weather they were able to zoom the screen to see the information larger. Participant 1 stated “being able to see the app is a big advantage” and “that’s why I tend not to use our Met Office app. Its unusable for me. So I just tend to go on the web”.</p>	<p>This is a positive benefit of web applications over native that was not discovered as part of the literature research. How impaired users are affected by applications was not considered and would be beneficial future research.</p>
<p>Touch events across browser are interesting... things like inertial scroll can be a bit inconsistent across different browser types (Participant 2).</p> <p>In reference to scroll issues, participant 2 stated that improved scrolling is something which is “baked in with the native”, And so when developing a web “you have to worry about things that are just free in a native app”. Their opinion was that they would rather deal with this and save by using the one-app-for-everybody concept of a PWA, rather than dealing with the huge amounts of money it costs to write native apps across different devices.</p>	<p>The inconsistency of scrolling is something noted by the negative user response of the open-ended question within the questionnaire and needs investigation. Participant 2 suggests that the cost of implementing multiple native applications across devices is significant, agreeing with the point made by (Wasserman, 2010, p.400) in the literature review.</p>

Interview Point Raised	Researcher's Analysis and Discussion
<p>Struggled with people picking up updates to the native app. Participant 2 stated that the Met Office is moving from an old native app to a new native app, and loads of people aren't coming off the old app and they want to turn it off, but are not able to because people still have it., they clarified that "if we want to change stuff we've got to keep backwards compatible versions of all our data feeds. For probably...ever. Or at least until we'll die. So there's massive advantages in my mind for a progressive one"</p>	<p>The agreement with the research by Sillars (2015, p.8), that the Met Office has issues with native app users updating is a positive bonus for PWA development. The added cost of maintaining backwards compatible services was not originally considered and would be another cost benefit for developing a PWA.</p>
<p>More of the native API's are available now on the web. Participant 2 stated that "unless you have a really specific for a native capability, I don't know why that [PWA development] shouldn't be considered first". They then went on to point that native API, like geolocation and the accelerometer, are available in a web context anyway.</p>	<p>Whilst this is in agreement with literature found, the lack of contacts, calendar and telephony data (Malovolta, 2016, p.2), may still cause issues. Lack of native functionality will be a main driver behind some organisations not developing PWAs in the short term.</p>
<p>Web Frameworks. In reference to using web frameworks, participant 2 stated that they "push you in a good architectural direction" and will "force a particular MVC style... which is generally a good practice to follow" They also stated that another reason for using web frameworks was they have a lot of teams developing; a framework ensures they are all using something similar, and if developers need to move team they don't have to completely learn a new way of building stuff.</p> <p>Participant 2 also suggested that "Angular 2 covers a lot of the basis you're [researcher] talking about there, from minifying the code, to tree-shaking, does a lot of that through the webpack plugins"</p>	<p>These findings agree with the point in the literature review that MVC frameworks are suitable when development is split amongst a team on a large, complex project (Wasserman, 2010, p.398).</p> <p>Development with the AngularJS framework has not been detrimental to the performance of PWA weather based on user feedback and therefore use of a framework is recommended.</p>

Interview Point Raised	Researcher's Analysis and Discussion
<p>Marketing the PWA. Participant 4 made comments in regards to marketing a PWA rather than a native app. They believed that if “you’ve already got essentially a captive audience who you know will follow a URL” then a PWA is viable, but if you are “reliant on the credibility or the apparent security of using an app store” then marketing could be an issue. There are “market segments that are cautious about this given certain warnings around only downloading approved apps from your secure play store”</p>	<p>The marketing of the PWA was not something originally covered in the literature review. PWA's can be shared easily by URL, but if users are reluctant to use the application if not from a secure play store this may be limiting.</p>
<p>Security. Participant 3 felt there may be some security concerns and that “people tend to view a web page more as something they can penetrate than they do an app. They’re going to denial of service, and stuff like that. That stuff is more hit than an app. Because its containerised” Participant 2 suggested some users may be uncertain given that “effectively it’s a webpage right, and people use a browser all the time, but they’ll feel like it’s an app, and not a webpage, but it’s not from an app store”</p>	<p>Whilst a PWA uses HTTPS, when in full screen mode this is not apparent and may cause users to be wary given they did not download the application from an approved store. Though a webpage is more discoverable and liable to attack, this is no more so for a PWA than existing websites and shouldn't be a concern.</p>
<p>Monetisation. Participant 2 raised some concerns over in-app payments stating “adverts, fine, you can drop that into a PWA fairly easily, you want to start taking payments for in-app, you’ve then got to find ways to support that. That users will feel comfortable with. They’ll trust Google and Apple with their money... until someone’s kind of cracked that particular nut, it will be harder to sort of get into your markets with it, a PWA”. Participant 2 went on to suggest there will be emerging API's or PayPal could be used. Participant 4 suggested possible integration with Apple pay.</p>	<p>The inclusion of a well-known web payment API's such as PayPal will likely be acceptable to most users given they are comfortable using these services already. Though whether this is the case would need further user research</p> <p>Integrating with Google or Apple web payment services such as the Payment Request API (Google, 2017) might be possible, but these are still in active development.</p>

Table 19 - Discussion of points raised during Met Office interview

6. Best practice recommendations

The following best practice recommendations are suggested to ensure PWA development is scalable and the application performant. These are based on the primary research collected; the experience gained from developing PWA Weather, the optimisation tests, and the end user and industry feedback.

- A JavaScript framework is recommended to facilitate an MVC architectural direction. This ensures maintainability, allows for easier unit testing and speeds up development.
- Ensure all code is minified and tree-shaked. In this study minification contributed to a 57% improvement in the time users receive the first meaningful content on 3G connections. Tree-shaking reduced code bundle size by 12.5%, and should be added as part of any continuous integration to protect against redundant code from being deployed.
- Combining an application shell with inline CSS and non-blocking JavaScript improves the initial paint of the application and is easily implemented; showing the core user interface quickly to the user. Non-blocking JS contributed to a 43.6% improvement in the time to First Paint.
- Combine CSS transforms and will-change to create animations that perform well on all mobile devices. There was an increase of 20 frames per second on the low-end device tested.
- Web workers were not found to be efficient when responding to user actions, so shift longer running tasks server-side if they slow the application.
- Where requests to server APIs are dependent upon one another, use GraphQL. This has been shown to reduce the time taken for data retrieval by up to 69% for this use case. GraphQL can prevent over-fetching and reduces the amount of server-side development required.

7. Conclusion

The objectives identified at the start of this paper have been met. A PWA was developed and optimisations were tested for their effectiveness. Research with users found the end performance and usability of PWA Weather to match or better that of existing native applications. From the interview with industry professionals it is notable that unless there is a definite need for native functionality core to the application, a PWA is a viable business and technical consideration for most cases. Whilst some concerns were raised by industry, in regards to in-app monetisation, marketing and the ability to perform intensive processing, they were generally positive about the prospect and the benefits a PWA can bring.

7.1 Limitations and future work

A limitation of this paper is the relatively small set of user responses to the questionnaire. A larger number of participants would provide a more in-depth analysis of how the PWA compares. Further research could look at the performance of the more recent version of the AngularJS framework (version 4 at time of writing) or look at comparing the performance impact of different frameworks on a PWA. Utilising web workers for multi-threading is an area that would benefit from significant additional research. Image processing is a limitation of JavaScript that has been discussed in the primary research of this paper. A maintainable solution that shows such intensive processing is efficiently possible within a web context would be a positive future development in terms of PWA's being able to replicate native performance.

8. References

- 1) Amazon Web Services. (2017). *AWS Elastic Beanstalk*. Available at: <https://aws.amazon.com/elasticbeanstalk/> (Accessed: January 2017)
- 2) AngularConnect. (2015). *Using Web Workers for more responsive apps – Jason Teplitz*. Available at: https://www.youtube.com/watch?v=Kz_zKXiNGSE (Accessed: November: 2016)
- 3) Cederlund, M. (2016). *Performance of frameworks for declarative data fetching: An evaluation of Falcor and Relay+GraphQL*. Master's Thesis. KTH Royal Institute of Technology, Stockholm, Sweden. Available at: <http://kth.diva-portal.org/smash/get/diva2:1045900/FULLTEXT01.pdf> (Accessed: November 2016)
- 4) Cerf, V.G. (2016). 'Apps and the web', *Communications of the ACM*. 59(2), p.7. doi: 10.1145/2872420
- 5) Cohen D. and Crabtree B. (2006). *Qualitative Research Guidelines Project*. Available at: <http://www.qualres.org/HomeUnst-3630.html> (Accessed: December 2016)
- 6) ComScore. (2016). *The 2016 U.S. Mobile App Report*. Available at: <http://www.comscore.com/Insights/Presentations-and-Whitepapers/2016/The-2016-US-Mobile-App-Report> (Accessed: October 2016)
- 7) Creswell J. W. (2013). *Research Design: Qualitative, Quantitative and Mixed Methods Approaches*. 4th edn. London: SAGE Publications
- 8) Dot, G., Martinez, A. and Gonzalez, A. (2015). 'Analysis and Optimization of Engines for Dynamically Typed Languages'. *2015 27th International Symposium on Computer Architecture & High Performance Computing (SBAC-PAD)*, IEEE, pp41-48
- 9) DoubleClick. (2016). *The need for mobile speed. Better user experiences, greater publisher revenue*. Available at: <https://www.doubleclickbygoogle.com/articles/mobile-speed-matters/> (Accessed: October 2016)
- 10)eSurvey (2017) *Create free online surveys & questionnaires with eSurvey creator*. Available at: <https://www.esurveycreeator.co.uk/> (Accessed: January 2017).
- 11)Facebook (2016). *GraphQL: Working Draft – October 2016*. Available at: <http://facebook.github.io/graphql/> (Accessed: November 2016)

- 12) Flirtman, M. (2016). *High performance mobile web: Best practices for optimizing mobile web apps*. Sebastopol, California: O'Reilly Media, Inc.
- 13) Freepix (2017). Free graphic resources for designers. Available at: <http://www.freepik.com/> (Accessed: January 2017)
- 14) Gaunt, M. (2016). *Service Workers: an Introduction*. Available at: <https://developers.google.com/web/fundamentals/getting-started/primers/service-workers> (Accessed: November 2016).
- 15) Google. (2016a). *Progressive Web Apps*. Available at: <https://developers.google.com/web/progressive-web-apps/> (Accessed: October 2016)
- 16) Google. (2016b). *Housing.com increases conversions and lowers bounce rate by 40% with new PWA*. Available at: <https://developers.google.com/web/showcase/2016/pdfs/housing.pdf> (Accessed: October 2016)
- 17) Google. (2016c). *Angular Universal*. Available at: <https://universal.angular.io/> (Accessed: December 2016)
- 18) GoogleChrome (2016). *Lighthouse – Auditing, performance metrics, and best practices for Progressive Web Apps*. Available at: <https://github.com/GoogleChrome/lighthouse> (Accessed: December 2016)
- 19) Google Chrome Developers (2016). *High performance web user interfaces - Google I/O 2016*. Available at: <https://www.youtube.com/watch?v=thNyy5eYfbc> (Accessed: November 2016)
- 20) Google (2017). *Payment Request API: an integration guide*. Available at: <https://developers.google.com/web/fundamentals/discovery-and-monetization/payment-request/> (Accessed: April 2017)
- 21) Grigorik, I. (2013a). *High Performance Browser Networking – Transport Layer Security (TLS)*. Available at: <https://hpbnp.co/transport-layer-security-tls/#https-everywhere> (Accessed: November 2016)
- 22) Grigorik, I. (2013b). *High Performance Browser Networking – Primer on Latency and Bandwidth*. Available at: <https://hpbnp.co/primer-on-latency-and-bandwidth/> (Accessed: November 2016)

- 23) Hewlett Packard Enterprise. (2015). *Failing to Meet Mobile App User Expectations: A Mobile User Survey - Dimensional Research*. Available at:
<http://go.saas.hp.com/techbeacon/apppulse-mobile-survey> (Accessed: October 2016)
- 24) Hwang, I. (2015). 'Design and implementation of cloud offloading framework among devices for web applications', *2015 12th Annual IEEE Consumer Communications & Networking Conference (CCNC)*, IEEE, pp.41-46
- 25) Johanan, J., Khan, T. and Zea, R. (2016). *Web Developer's Reference Guide*. Birmingham, UK: Packt Publishing Ltd.
- 26) Kearny, M. (2016). *Measure Performance with the RAIL Model*. Available at:
<https://developers.google.com/web/fundamentals/performance/rail> (Accessed October 2016).
- 27) Malavolta, I. (2016). 'Beyond native apps: web technologies to the rescue! (keynote)', *In Proceedings of the 1st International Workshop on Mobile Development*, ACM, New York, NY, USA, 1-2
- 28) Met Office (2016a). *Met Office Mobile Weather*. Available at:
<http://www.metoffice.gov.uk/mobile/> (Accessed: November 2016)
- 29) Met Office (2016b). *Met Office Weather App*. Available at:
<http://www.metoffice.gov.uk/services/mobile-digital-services/weather-app> (Accessed: November 2016)
- 30) Met Office (2017) Met Office DataPoint. Available at:
<http://www.metoffice.gov.uk/datapoint> (Accessed: January 2017)
- 31) NG-Conf. (2016). *Angular 2 Universal Patterns - Jeffrey Whelpey, Patrick Stapleton*. Available at: https://www.youtube.com/watch?v=TCj_oC3m6_U (Accessed: November 2016)
- 32) Oh, J. and Moon, S. (2015). 'Snapshot-based loading-time acceleration for web applications', *2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, IEEE, San Francisco, CA, 2015, pp. 179-189.
- 33) Osmani, A. and Gaunt, M. (2015). *Instant Loading Web Apps with an Application Shell Architecture*. Available at: <https://developers.google.com/web/updates/2015/11/app-shell> (Accessed October 2016)

- 34)Rajeev B.V. and Bakula, K. (2015). 'A developer's insights into performance optimizations for mobile web apps', *Advance Computing Conference (IACC), 2015 IEEE International*, Bangalore, 2015, pp. 671-675. doi: 10.1109/IADCC.2015.7154791
- 35)Rendrjs. (2016). *Rendr*. Available at: <https://github.com/renderjs/render> (Accessed: December 2016)
- 36)Russell, A. (2016). *What, Exactly, Makes Something A Progressive Web App?* Available at: <https://infrequently.org/2016/09/what-exactly-makes-something-a-progressive-web-app/> (Accessed: November 2016)
- 37)Sillars, D. (2015). *High performance Android apps: Improve ratings with speed, optimizations, and testing*. Sebastopol, California: O'Reilly Media, Inc.
- 38)SimilarTech. (2016). *Technologies Market Share - JavaScript*. Available at: <https://www.similartech.com/categories/javascript> (Accessed: November 2016)
- 39)Sommerville, I. (2016). *Software Engineering*.10th edn. London: Pearson Education Limited.
- 40)Strimpel, J and Najim, M. (2016). *Building Isomorphic JavaScript Apps: From concept to implementation to real-world solutions*. Sebastopol, California: O'Reilly Media, Inc
- 41)Syromiatnikov, A and Weyns, D. (2014). 'A Journey through the Land of Model-View-Design Patterns', *2014 IEEE/IFIP Conference on Software Architecture (WICSA)*, IEEE, Sydney, NSW, 2014, pp. 21-30
- 42)Wasserman, A. I. (2010). 'Software engineering issues for mobile application development', *In Proceedings of the FSE/SDP workshop on Future of software engineering research*, ACM, New York, NY, USA, pp.397-400
- 43)Wenzel, M and Meinel, C. (2015) 'Parallel network data processing in client side JavaScript applications,' *2015 International Conference on Collaboration Technologies and Systems (CTS)*, Atlanta, GA, 2015. IEEE, pp.140-147. doi: 10.1109/CTS.2015.7210414
- 44)Willocx, M., Vossaert, J. and Naessens, V. (2016). ' Comparing performance parameters of mobile app development strategies', *In Proceedings of the International Conference on Mobile Software Engineering and Systems*, ACM, New York, NY, USA, pp.38-47

- 45)W3C (2015a). *Web Workers - W3C Working Draft 24 September 2015*. Available at:
<https://www.w3.org/TR/workers/> (Accessed: November 2016)
- 46)W3C (2015b). *W3C DOM4 - W3C Recommendation 19 November 2015*. Available at:
<https://www.w3.org/TR/dom/> (Accessed: November 2016)
- 47)W3C (2016a). *Web App Manifest - W3C Working Draft 07 November*. Available at:
<https://www.w3.org/TR/2016/WD-appmanifest-20161107/> (Accessed: November 2016)
- 48)W3C (2016b). *CSS Will Change Module Level 1*. Available at:
<https://www.w3.org/TR/css-will-change/> (Accessed: November 2016)

9.3 Appendix 3: Requirements Specification & Wireframe

Purpose

Production of a Progressive Web Application (PWA) that offers forecast weather data targeted at the public. Type of data offered to mirror that currently delivered by the Met Office mobile website (<http://www.metoffice.gov.uk/mobile/>) and native application. The purpose of the application is to test the ability of PWA's to offer a comparable alternative to native platforms.

System Overview

Weather Data for the application will be acquired using the Met Office public API DataPoint (<http://www.metoffice.gov.uk/datapoint/product>). The application will make use of a text-based search functionality to provide weather forecasts for a specific location. This will be presented in the form of a table showing weather forecast information at 3 hourly intervals for the next 5 days, as well as summary text from forecasters. The AngularJS framework will be used for maintainability; ensuring the application will have the architecture to grow and develop to the scale required in industry. AngularJS was chosen due to being the current leader of MVC frameworks within the top 10k most visited sites.

The application will include the following key concepts of PWA's:

- Service Worker for offline functionality and caching.
- Application shell architecture.
- Home screen icon, full-screen viewing and splash-screen.
- Push notifications.

The following areas will be investigated for speed improvements after development:

- Async loading.
- Minification and Tree Shaking.
- CSS based performance improvements (pointer-events / opacity, transforms / will-change).
- Web Workers and separate UI Thread.
- Server-side rendering.
- GraphQL for the data service

Specific Requirements

Application to be hosted on a secure HTTPS domain as part of PWA requirements. This will be achieved through the free tier of Amazon Web Service (<https://aws.amazon.com/>).

Project Timetable

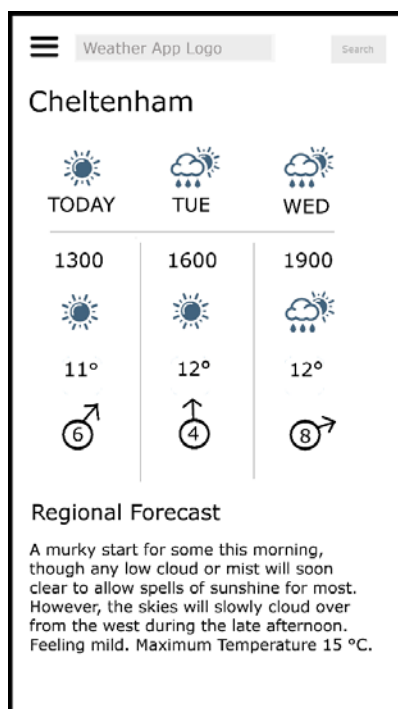
Estimated hours to complete initial application (minus optimisations):

Basic Structure of Application	10
Service to pull weather data	10
Displaying data in table format	7
Text Search	7
Add to home screen install banner, splash screen and icon	4
Service worker and basic offline caching	4
Push notifications	6
Total:	48

Proposing to build the initial PWA by January 2017. Demo of the product and test results to be given at a seminar at Met Office HQ, Exeter in late February / early March 2017.

Draft Wireframe

A preliminary wireframe has been produced showing the proposed look of the application,



though is subject to change once development begins:

A search button will make a text input visible & animate from the top of the screen, where a new location can be entered and searched for. An additional burger menu animates a side navigation bar that will contain recently searched for destinations. The top row of the forecast table gives an overview of the weather for the next 5 days (can be scrolled horizontally). Detailed information is then displayed showing weather type, temperature and a wind speed/direction icon for each of the 3-hourly time-steps. Regional forecast text will also be displayed beneath the forecast table.

9.4 Appendix 4: Lighthouse PWA Report for pwaweather.xyz (Dev branch)

3/18/2017

Lighthouse report: <https://www.pwaweather.xyz/>

88
100

Progressive Web App

These audits validate the aspects of a Progressive Web App. They are a subset of the [PWA Checklist](#).

✓ App can load on offline/flaky connections ▾

Ensuring your web app can respond when the network connection is unavailable or flaky is critical to providing your users a good experience. This is achieved through use of a [Service Worker](#).

- ✓ Registers a Service Worker ?
- ✓ Responds with a 200 when offline ?

! Page load performance is fast ▾

Users notice if sites and apps don't perform well. These top-level metrics capture the most important perceived performance concerns.

- 34 First meaningful paint: **4920.3ms** (target: 1,600ms) ?
- 46 Perceptual Speed Index: **5917** (target: 1,250) ?
 - First Visual Change: **5680ms**
 - Last Visual Change: **6743ms**
- 88 Estimated Input Latency: **61.4ms** (target: 50ms) ?
- 41 Time To Interactive (alpha): **5654.6ms** (target: 5,000ms) ?

3/18/2017

Lighthouse report: <https://www.pwaweather.xyz/>

✓ Site is progressively enhanced ▾

Progressive enhancement means that everyone can access the basic content and functionality of a page in any browser, and those without certain browser features may receive a reduced but still functional experience.

- ✓ Contains some content when JavaScript is not available ?

! Network connection is secure ▾

Security is an important part of the web for both developers and users. Moving forward, Transport Layer Security (TLS) support will be required for many APIs.

- ✓ Uses HTTPS ?
- ✗ Redirects HTTP traffic to HTTPS ?

✓ User can be prompted to Add to Homescreen ▾

While users can manually add your site to their homescreen in the browser menu, the [prompt \(aka app install banner\)](#) will proactively prompt the user to install the app if the below requirements are met and the user has visited your site at least twice (with at least five minutes between visits).

- ✓ Registers a Service Worker ?
- ✓ Manifest exists ?
- ✓ Manifest contains start_url ?
- ✓ Manifest contains icons at least 144px: **found sizes: 144x144, 192x192**
- ✓ Manifest contains short_name ?

blob:chrome-extension://blipmcconikpinedfhnrmjammmfjpmptjku/1a3ba080-20c2-48fe-a5fc-8770d5c7b0be

2/8

✓ Installed web app will launch with custom splash screen ▾

A default splash screen will be constructed, but meeting these requirements guarantee a high-quality and customizable [splash screen](#) the user sees between tapping the home screen icon and your app's first paint.

- ✓ Manifest exists ?
- ✓ Manifest contains name ?
- ✓ Manifest contains background_color ?
- ✓ Manifest contains theme_color ?
- ✓ Manifest contains icons at least 192px: **found sizes: 192x192** ?

✓ Address bar matches brand colors ▾

The browser address bar can be themed to match your site. A theme-color [meta tag](#) will upgrade the address bar when a user browses the site, and the [manifest theme-color](#) will apply the same theme site-wide once it's been added to homescreen.

- ✓ Manifest exists ?
- ✓ Has a <meta name="theme-color"> tag: **#00728C**
- ✓ Manifest contains theme_color ?

✓ Design is mobile-friendly ▾

Users increasingly experience your app on mobile devices, so it's important to ensure that the experience can adapt to smaller screens.

- ✓ Has a <meta name="viewport"> tag with width or initial-scale ?
- ✓ Content is sized correctly for the viewport ?

Best Practices

We've compiled some recommendations for modernizing your web app and avoiding performance pitfalls. These audits do not affect your score but are worth a look.

✓ Using modern offline features ▾

- ✓ Avoids Application Cache ?
- ✓ Avoids WebSQL DB ?

! Using modern protocols ▾

- ✓ Uses HTTPS ?
- ✗ Uses HTTP/2 for its own resources: **10 requests were not handled over h2** ?

▼ View details

URL

<https://www.pwaweather.xyz/>

<https://www.pwaweather.xyz/assets/search.png>

<https://www.pwaweather.xyz/app-shell.min.js>

<https://www.pwaweather.xyz/app.bundle.min.js>

<https://www.pwaweather.xyz/proxy?url=http://datapoint.metoffice.gov.uk/public/data/val/wxfcs/all/json/310047?res=daily&key=0418f3e9-cdb2-4ad8-f131-41d014110000>

<https://www.pwaweather.xyz/proxy?url=http://datapoint.metoffice.gov.uk/public/data/txt/wxfcs/regionalforecast/json/513?key=0418f3e9-cdb2-4ad8-f131-41d014110000>

<https://www.pwaweather.xyz/proxy?url=http://datapoint.metoffice.gov.uk/public/data/val/wxfcs/all/json/310047?res=3hourly&key=0418f3e9-cdb2-4ad8-f131-41d014110000>

<https://www.pwaweather.xyz/proxy?url=http://datapoint.metoffice.gov.uk/public/data/val/wxfcs/all/json/sitelist?key=0418f3e9-cdb2-4ad8-f131-41d014110000>

https://www.pwaweather.xyz/assets/wind_arrow_sprites.png

https://www.pwaweather.xyz/assets/weather_icon_sprites.png

Accessibility

- ✓ Element aria-* attributes are allowed for this role ?
- ✓ Elements with ARIA roles have the required aria-* attributes ?
- ✓ Element aria-* attributes are valid and not misspelled or non-existent. ?
- ✓ Element aria-* attributes have valid values ?
- ✓ Background and foreground colors have a sufficient contrast ratio ?
- ✓ Every image element has an alt attribute ?
- ✓ Every form element has a label ?
- ✓ No element has a tabindex attribute greater than 0 ?

Other

- ✓ Manifest's short_name won't be truncated when displayed on homescreen ?
- ✓ Manifest's display property is set: **standalone** ?

Performance

These encapsulate your app's performance.

Unused CSS rules: Potential savings of 2 KB (~10ms) ?

▼ View details

URL	Unused Rules	Original	Potential Savings
blob:chrome-extension://blipmddonikpinefhnrmjammfjpmptjku/1a3ba080-20c2-48fe-a5fc-8f70ddc7b0be			

blob:chrome-extension://blipmddonikpinefhnrmjammfjpmptjku/1a3ba080-20c2-48fe-a5fc-8f70ddc7b0be

6/8

<i>inline</i>			
ul[_ngcontent-wen-1] { width: 100%; padding: 0; overflow-x: auto; margin: 0; ... } ...	50	2 KB	2 KB 66%

Unoptimized images: Potential savings of 59 KB (~290ms) ?

▼ View details

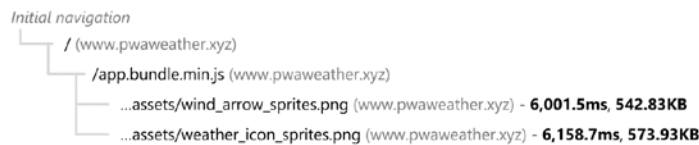
URL	Original	WebP Savings	JPEG Savings
...assets/weather_icon_sprites.png	106 KB	36 KB 34%	--
...assets/wind_arrow_sprites.png	75 KB	24 KB 31%	--

Oversized Images ?

Critical Request Chains: 2 ?

Longest chain: **6,158.7ms** over **3** requests, totalling **106.44KB**

▼ View critical network waterfall



Render-blocking Stylesheets ?

Render-blocking scripts ?

100 Avoids enormous network payloads: Total size was 820 KB (target: 1,600 KB) ?

▼ View details

blob:chrome-extension://blipmddonikpinefhnrmjammfjpmptjku/1a3ba080-20c2-48fe-a5fc-8f70ddc7b0be

7/8

URL	Total Size	Transfer Time
/app.bundle.min.js	465 KB	2,280ms
/proxy?url=...	161 KB	790ms
...assets/weather_icon_sprites.png	106 KB	520ms
...assets/wind_arrow_sprites.png	75 KB	370ms
/app-shell.min.js	6 KB	30ms
/	2 KB	10ms
/proxy?url=...	1 KB	10ms
/proxy?url=...	1 KB	10ms
...assets/search.png	1 KB	10ms
/proxy?url=...	1 KB	10ms

100 Avoids an excessive DOM size: **119 nodes** (target: 1,500 nodes) ?

▼ More information

Total DOM Nodes	DOM Depth	Maximum Children
119 <small>target: < 1,500 nodes</small>	14 <small>target: < 32</small>	12 <small>target: < 60 nodes</small>

0 User Timing marks and measures: **0** ?

Fancier stuff

blob:chrome-extension://blipmööonikpinefhnmmjamfjpmptjku/1a3ba080-20c2-48fe-a5fc-8f70ddc7bebe

8/8

9.5 Appendix 5: Questionnaire

Application Performance Survey - PWA Weather

Page 1

This survey is designed to gain user feedback on the performance of a mobile application, PWA Weather. The application allows for the searching of a weather forecast by UK location. **This survey requires access to a smartphone to be completed. Prior use of an existing weather application is also required.**

Please provide the name(s) of any weather applications you have used on your smartphone: *

Please name the weather application you use most frequently: *

Please provide the make and model of your smartphone (if unsure leave field empty):

How would you rate the level of your digital skills (your ability to use computers, smartphones, apps, the internet etc..)?

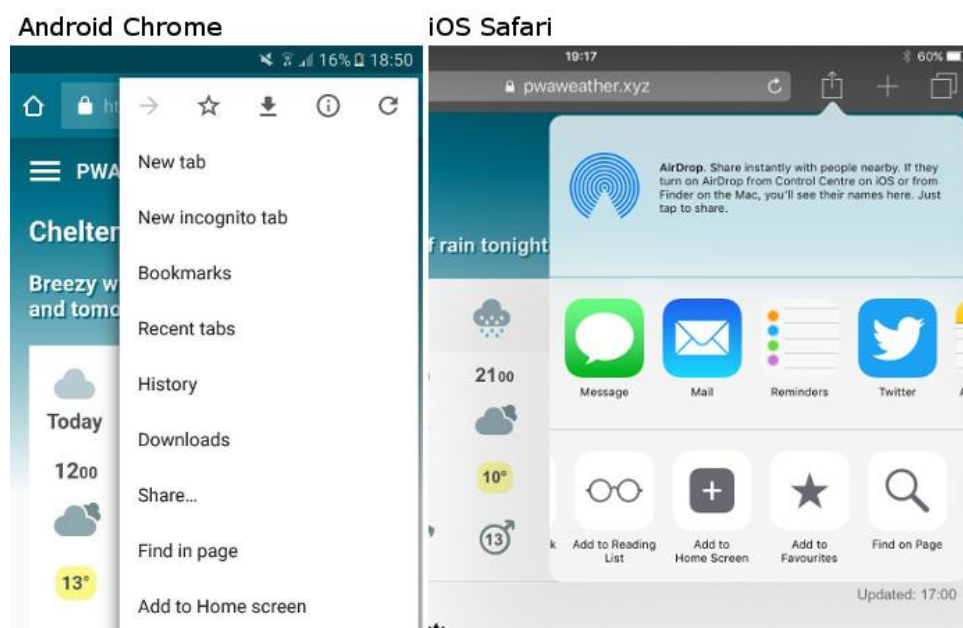
	Novice	Advanced Beginner	Competent	Proficient	Expert
Level	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Page 2

This survey involves testing PWA Weather on your smartphone. Please follow the steps below to add the application to your device:

1. Open a browser on your smartphone and visit the url: pwaweather.xyz
2. Using your browser menu, add the application to the home screen of your device (see screenshots below for Android and iOS guidance on how to do this)
3. Close your browser

Guidance images for adding PWA Weather to your device



If using Chrome on Android, select the browser menu using the button with 3-vertical dots in the top right corner. A menu should appear (see above, left). Select 'Add to Home Screen' and confirm any additional messages.
If using Safari on iOS, select the share menu (see above, right), click 'Add to Home Screen' and confirm any additional messages.

Find and select the PWA Weather icon on your home screen. Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather loads? *

- ☐ Not as fast
- ☐ The same
- ☐ Faster
- ☐ Don't know

Page 3

Use PWA Weather to search for a forecast. Compared to your ‘frequently used weather application’, how would you rate the speed that PWA Weather returns a forecast? *

- ☐ Not as fast
- ☐ The same
- ☐ Faster
- ☐ Don't know

Page 4

Click the burger menu in the top left corner and select any one of the previous locations.

Compared to your ‘frequently used weather application’, how would you rate the performance of animations within PWA Weather? *

- ☐ Not as smooth
- ☐ The same
- ☐ Smoother
- ☐ Don't know

Page 5

If you have any other comments about the performance of PWA Weather, please provide them below:

9.6 Appendix 6: Questionnaire Results

All questionnaire results. 6 participants failed to complete all questions. Responses have been greyed out to identify this.

Date and time	Participation status	1. Please provide the name(s) of any weather applications you have used on your smartphone:	2. Please name the weather application you use most frequently:	3. Please provide the make and model of your smartphone (if unsure leave field empty):	4. How would you rate the level of your digital skills (your ability to use computers, smartphones, apps, the internet etc..)?	5. Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather loads?	6. Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather returns a forecast?	7. Compared to your 'frequently used weather application', how would you rate the performance of animations within PWA Weather?	8. If you have any other comments about the performance of PWA Weather, please provide them below:
22/03/2017 14:52	participated but not yet completed	Met Office; BBC Weather; Accuweather	Met Office	Sony Xperia Z2	5				
22/03/2017 17:02	participated but not yet completed	Met Office; BBC; Weather.com	Met Office	iPhone 7	5				
23/03/2017 17:27	participated but not yet completed	BBC Weather	BBC Weather	Sony Xperia XA	4				
23/03/2017 21:54	participated but not yet completed	BBC Weather	Bbc weather	HTC one m9	4				
24/03/2017 05:33	participated but not yet completed	Met Office App	Met office App	LG	5				
24/03/2017 10:09	participated but not yet completed	BBC, Met Office, Weather Bug, Google App	Google	Nexus 6	4				
19/03/2017 00:17	participated and completed	BBC Weather, Met Office Weather and AccuWeather	BBC Weather	Samsung S6	5	The same	Faster	The same	Seemed just as fast and easy to use as the BBC Weather application. The process of adding the app to my home screen was faster than if I had installed the application from Google Play store.
19/03/2017 21:47	participated and completed	BBC weather ; Met office	BBC weather	Acatel one touch idol 3	4	The same	Faster	Smoother	
19/03/2017 22:04	participated and completed	BBC weather	BBC weather	Moto G	3	The same	The same	Not as smooth	
22/03/2017 10:47	participated and completed	Met Office; Apple Weather; WeatherGods	Met Office	iPhone 6S	5	Faster	Faster	Smoother	A fantastically fast and beautiful app. Well done :)

Date and time	Participation status	1. Please provide the name(s) of any weather applications you have used on your smartphone:	2. Please name the weather application you use most frequently:	3. Please provide the make and model of your smartphone (if unsure leave field empty):	4. How would you rate the level of your digital skills (your ability to use computers, smartphones, apps, the internet etc..)?	5. Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather loads?	6. Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather returns a forecast?	7. Compared to your 'frequently used weather application', how would you rate the performance of animations within PWA Weather?	8. If you have any other comments about the performance of PWA Weather, please provide them below:
22/03/2017 11:32	participated and completed	Met Office; BBC weather	Met Office	iPhone 5S	5	Faster	Faster	Smoother	The load times of the forecast data & speed of search results is really impressive
22/03/2017 12:41	participated and completed	Met Office App; BBC Weather	Met Office	Sony Xperia SP	5	Faster	The same	Smoother	The search functionality was slow to replicate typing, this could be an issue with my keyboard or the application
22/03/2017 12:52	participated and completed	- TAF/METAR; - SkyDemon; - Meteo-Earth	TAF/METAR	iPhone SE	4	The same	The same	The same	- Nice little weather application with simple, clear design; - Good concept to compare PWA vs Native app; - Speed/performance is comparable to Native apps, which is nice to see; - 'Install-able' feel of PWA is a great improvement over standard web app
22/03/2017 17:44	participated and completed	Weather Forecast: UK; Met Office Weather Forecast	Met Office Weather Forecast	Samsung Galaxy Mega 6.3	5	The same	Faster	Smoother	What's there is great: For a limited-time development it's sufficiently 'rich' to be a good showcase and keep my interest for future possibilities.; ; The sideways scrolling of weather symbols etc is great: symbols cut in half on the edge of the window make it really clear that the user should scroll.
22/03/2017 22:18	participated and completed	AccuWeather; Netweather.tv weather radar website; Lightningmap.org website	Netweather.tv weather radar website	Samsung s6	5	The same	Faster	Smoother	I have experienced no problems with the performance.
22/03/2017 22:40	participated and completed	MSN Weather; Met Office	MSN Weather	Nokia Lumia	4	The same	The same	The same	I like having the extra descriptive information below the quick reference icons. It relays more information without being cluttered. Would recommend

Date and time	Participation status	1. Please provide the name(s) of any weather applications you have used on your smartphone:	2. Please name the weather application you use most frequently:	3. Please provide the make and model of your smartphone (if unsure leave field empty):	4. How would you rate the level of your digital skills (your ability to use computers, smartphones, apps, the internet etc..)?	5. Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather loads?	6. Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather returns a forecast?	7. Compared to your 'frequently used weather application', how would you rate the performance of animations within PWA Weather?	8. If you have any other comments about the performance of PWA Weather, please provide them below:
22/03/2017 22:47	participated and completed	Default iphone weather app	iphone weather app	iphone SE	4	The same	The same	Don't know	
22/03/2017 23:45	participated and completed	BBC weather ; Accuweather	Accuweather	Samsung Galaxy s3	3	Not as fast	Faster	Smoother	A bit slow for the initial load, but very quick performance once loaded.
23/03/2017 01:47	participated and completed	Yahoo Weather; Google News & Weather ; Google Now	Google Now	Huawei Nexus 6P	5	Not as fast	Faster	The same	
23/03/2017 16:26	participated and completed	met office; weather.com	weather.com	Motorola Moto G4	3	The same	The same	The same	
23/03/2017 16:39	participated and completed	BBC ; Metoffice; Google	Google	Motorola	5	Not as fast	Not as fast	The same	Couldn't actually use the search, nothing happened.
23/03/2017 17:23	participated and completed	Apple weather app	Apple weather app	Apple iPhone 6s	4	Faster	Faster	The same	I prefer the layout of the pwa ; But I like being able to see an hour by hour weather report like the apple one has
23/03/2017 17:26	participated and completed	AccuWeather; Google (weather card/widget on Google screen)	AccuWeather	OnePlus 3	5	The same	Faster	Not as smooth	Regarding animations; can't actually see any animation at all unless it's the very slight flicker when you view a new forecast. Too fast to really notice.
23/03/2017 17:27	participated and completed	bbc weather	bbc weather	Samsung galaxy s6	5	The same	The same	Not as smooth	
23/03/2017 17:47	participated and completed	Weather	Weather	iPhone 7	4	The same	The same	The same	
23/03/2017 17:56	participated and completed	Sony Xperia Weather	Sony Xperia Weather	Sony Xperia XA	5	Faster	Faster	The same	I like Applications that are simple but effective, and this one does just that. Plus, I like the convenience of having it available on my phone without having to install it.

Date and time	Participation status	1. Please provide the name(s) of any weather applications you have used on your smartphone:	2. Please name the weather application you use most frequently:	3. Please provide the make and model of your smartphone (if unsure leave field empty):	4. How would you rate the level of your digital skills (your ability to use computers, smartphones, apps, the internet etc..)?	5. Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather loads?	6. Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather returns a forecast?	7. Compared to your 'frequently used weather application', how would you rate the performance of animations within PWA Weather?	8. If you have any other comments about the performance of PWA Weather, please provide them below:
23/03/2017 18:02	participated and completed	Google (data from weather.com; web app); Met Office Weather Forecast;; BBC Weather,	Google	Samsung Galaxy S5	5	The same	The same	The same	
23/03/2017 19:24	participated and completed	'Weather' Apple; BBC Weather	Weather (Apple)	iPhone 5c	4	Faster	Faster	The same	Great product
23/03/2017 21:01	participated and completed	Weather (iOS); BBC Weather	Weather (iOS)	iPhone 6S	5	Not as fast	The same	Not as smooth	Some features are laggy, can definitely tell it's a web-app. Header bar doesn't stick to the top all of the time. Parts of the application will jump rather than smoothly scroll. Some parts don't seem as responsive to touch compared to native applications.
23/03/2017 22:23	participated and completed	Yahoo!'s weather app; BBC's weather app; Apple's weather app	Bbc	Apple iPhone 6S 128gb	5	The same	Faster	Smoother	I believe the performance is on par, if not better than the current applications available. I like the simplicity to give you the details fast and efficiently. Thus, creating a smooth experience for the end user.
24/03/2017 10:28	participated and completed	BBC Weather; YR; Rain Alarm	BBC Weather	Motorola Moto G	4	Not as fast	Faster	Smoother	
24/03/2017 12:15	participated and completed	Only the Met Office one	Met Office	Xperia Z3	5	Faster	Faster	Smoother	Performance is outstanding. Intrigued to find that it works without a connection. Presume the locations are precached and so is the weather?; Still very fast, light and basic.
24/03/2017 13:29	participated and completed	BBC weather;; AccuWeather;; HTC weather	AccuWeather	Wileyfox swift 2 plus	5	Faster	The same	Smoother	

Date and time	Participation status	1. Please provide the name(s) of any weather applications you have used on your smartphone:	2. Please name the weather application you use most frequently:	3. Please provide the make and model of your smartphone (if unsure leave field empty):	4. How would you rate the level of your digital skills (your ability to use computers, smartphones, apps, the internet etc..)?	5. Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather loads?	6. Compared to your 'frequently used weather application', how would you rate the speed that PWA Weather returns a forecast?	7. Compared to your 'frequently used weather application', how would you rate the performance of animations within PWA Weather?	8. If you have any other comments about the performance of PWA Weather, please provide them below:
24/03/2017 14:40	participated and completed	BBC weather App; Met Office weather ; Yahoo	Met Office mobile website	Moto G (1st Gen)	5	The same	Faster	Smoother	
25/03/2017 21:26	participated and completed	Met office; iPhone weather	Met office	iPhone 5	3	Not as fast	The same	The same	Met office has weather each hour where as this is every three. ; I prefer to scroll up and down rather than left to right to see how the days weather progresses.
26/03/2017 16:26	participated and completed	AccuWeather	AccuWeather	Xperia XZ	5	The same	Faster	Smoother	Very good app, shows the relevant weather accurately. Would definitely use the finished app!
27/03/2017 16:53	participated and completed	BBC Weather	BBC Weather	HTC M8	4	The same	The same	Smoother	Looks good. I would like more features in future.

9.7 Appendix 7: Met Office Interview Transcript

Met Office Interview Transcript

21st March 2017

Introduction: Began by researcher presenting an overview of the papers' secondary research, the developed PWA and primary findings to date. Full PowerPoint presentation is available within Appendix.5. The seminar was attended by 11 Met Office employees based within the applications team, responsible for development of external, customer facing applications. 5 main individuals contributed as part of the discussion, job titles and descriptions as follows:

Participant 1: - Developing in house bespoke applications for the commercial and public weather sector, using web technologies such as HTML5, JavaScript, AngularJS, Adobe Flex, Java, Oracle, and web services.

Participant 2: - Accountable for the delivery of customer facing weather services to the web, including the public weather service and phone app, as well as sector specific services such as road, rail and aviation. Including planning, resourcing of teams, business liaison, negotiating with suppliers and recruitment.

Participant 3: - responsible for test planning, communicating with stakeholders on the progress of products and for producing exit reports. Record defects appropriately using tools including HP Quality Centre and JIRA. Host defect review meetings.

Participant 4: - develop and support decision tools, collaborate with colleagues and clients to integrate data, systems and visualisations to support confident decision-making in weather-sensitive operations.

Participant 5: - Working as one of two front-end developers within an agile team, building Angular 2 applications. Lead role in shaping the direction in which to take the front end, while working closely with embedded designers, testers and the Technical Lead.

Transcript begins at the point of first interview question following the presentation:

Researcher: “Is there any extra functionality that you might want to put in, maybe you’ve already got in the Met Office native app, that might be a problem if we were to do it with PWAs instead?”

Participant 1: “One of the things they’re doing with the native app is with the map they’re putting layers on top. So they’re doing like rain layers, it’s one of the things I looked at because I wanted to do what they’re doing on our website... we can’t do it because we haven’t got access to the internal hardware. What they’re doing is they’re stitching images together. And then they’re doing colour state changes and stuff, loads of stuff on-the-fly, which if you did it not on a native app it would really kill the application, but because it’s a native app they’ve got more access to the lower level operating systems. They can harness more of the power of the device, but you can’t do that on the web side. There’s kind of performance things like that.”

Researcher: “Is that more of a lack of threads issue?”

Participant 1: “Yeah, I think so...cause obviously you’re multi-threaded when you’re doing native apps so you can do more stuff simultaneously, but you can’t do that on the web. But whether you can move some of that to the backend and do it, or use a web worker.”

Researcher: “That’s something a web worker might be able to do, but it would be a lot of work to figure that out.”

Participant 2: “Yeah...what they’re doing with the animation of rain in the app at the moment, is they’re taking the image, but they’re taking 8 images I think, at least, and compressing them into one image, and they’re using... for every pixel they’re using the RGBA values, to decide what the colour of an actual pixel is...because the actual colour palette is like 8 different colours. But RGBA gives them a lot more than that. So by using the first two parts of each number, they can define 8 different...effectively 8 different steps for every image. So then what you’ve got to do is you’ve got to read that image, each pixel value, each RGB value, and then redraw that onto the canvas. Now whether you could do that efficiently with a JavaScript engine on a browser, don’t know...might be able to. I think that will improve as the hardware improves. That’s not going to be as performant as a native app would do, but it would be worth having a look. You can get surprised by some things. But that certainly feels like an area that the native might have an advantage. Just the processing power that it can chuck at this stuff. Most of the time I don’t, certainly for a weather style app, I don’t see any reason why we wouldn’t have been doing this. I wanted

to do this the first time around, but I wasn't in charge of the app, so. Performance-wise most of what we're doing is just data volume, data sort of retrieval it's not overly complex in terms of sort of running big algorithms, across it, so I don't see any reason why that would be a problem. Can server side any of that anyway."

Participant 1: "Another big plus for the web-side as opposed to native, is on native you can't do that [demonstrates zooming in on the screen], which is obviously I'm a niche case [participant 1 has a visual impairment], but you know, being able to see the app is a big advantage. That's why I tend not to use our Met Office app. Its unusable for me. So I just tend to go on the web...and easily look at something. So you can just zoom it and not have to worry."

Participant 3: "But people have sort of moved away from expecting to go to a webpage, that's the trouble now. They just expect to click a tile and everything happens."

Researcher: "Once you've added that to your home screen it will be a tile the same as any other application, so it will open up full-screen"

Participant 4: "It does, I've just seen it and it works"

Participant 2: "Thrills of a live demo"

Participant 1: "When you visit it it'll ask you if you want to add it to your home screen."

Participant 3: "Ah ok I didn't get that, I probably didn't look for it."

Researcher: "You normally have to visit it twice because Chrome have put in a restriction so you don't get bombarded with the requests"

Participant 1: "You've got an inferior iPhone anyway. Get a proper phone"

Researcher: "That's the one current issue with iPhones is that the Service Worker isn't supported, so you can't get any of the offline stuff, unless you were to use AppCache, but that's a bit old now."

Participant 3: "It doesn't like scrolling on the iPhone either. No its sort of when you scroll... you lose the top."

Researcher: "Is that added as an app."

Participant 3: "No, this is just going from the web page."

Participant 2: “We’ve found when we’re just doing standard web apps, that touch events across browser are interesting...things like inertial scroll can be a bit inconsistent across different browser types, which obviously you get baked in with the native, so at least you don’t have to worry about stuff like that. So occasionally you have to worry about things that are just free in a native app. But I think, personally, I would rather worry about those and save on the one-app-for-everybody concept, rather than the huge amounts of money it costs to write native apps across different devices. We certainly have struggled with people picking up updates, that has absolutely affected us, we’re moving from an old app to a new app, and loads of people aren’t coming off the old app and we want to turn it off. We can’t turn it off because they’ve still got it.”

Participant 1: “You haven’t seen that thread on the...”

Participant 3: “I must admit I’m one that hasn’t uninstalled it cause I want to see how they make it die nicely”

Participant 2: “Well they’re going to struggle to make it die nicely, and yeah updates and stuff like that, if we want to remove feeds, if we want to change stuff we’ve got to keep backwards compatible versions of all our data feeds. For probably...ever. Or at least until we’ll die. So there’s massive advantages in my mind for a progressive one. So unless you have a...the way I would look at it unless you have a really specific for a native capability, I don’t know why that shouldn’t be considered first. We should be, I think they’re at a stage now, they’re mature enough to do it. And they’re opening up more and more of the API, of the actual native API’s to these things anyway. So you couldn’t use to get locations, but now you can. You couldn’t use to think get to the accelerometer, but you can on some. And all that sort of stuff so, I mean actually our app doesn’t need some of that stuff, but when you do.”

Researcher: “So one of the other questions I have on there, what about doing it without a framework, to kind of get less code in there...is there any reason to necessarily use Angular or anything like that?”

Participant 2: “No...I think you use a framework because it makes your life easier. If it doesn’t make your life easier than don’t use it. In the same way as you pick one over another. It gets quite a personal preference. Depends how big your software house is. As we’ve got a lot of teams doing stuff we are trying to make it so we’re all using something similar, so that if we need to switch one dev out to another team, they don’t have to

completely learn a new way of building stuff. In terms of why you build one app in one framework, that's just up to you. You know if you're picking one now you might not even pick Angular today, in 6 months you might pick something else. 2 years ago you wouldn't have picked it. Who knows, it's a very fast changing part of the world."

Participant 1: "Angular 2 covers a lot of the basis you're talking about there, from minifying the code, to tree-shaking, does a lot of that through the webpack plugins. You can do, say the offline template building so you can go straight with mocked up first page before it gets data, so you've got something to show straight away. Angular 2 has a lot of that stuff ready to go."

Participant 2: "I mean one of the things that a JavaScript framework will help with is it kind of, it opinionates how you build something, it opinionates how you architect something, so it will force a particular MVC style or something like that on you. Which is generally a good practice to follow. If you do it yourself, you could do some whacking great monolith, and that's your bag but, probably not the right way to write it. A framework will tend to push you in a good architectural direction. Can you do it without, course you can yeah, and at times you will want to, because there are somethings you really want to get into the native JavaScript and not the framework because it will be more performant. And you could have a look at that, but it's a bit case by case really."

Researcher: "Next questions...Some of its already kind of been covered, like hardware or software features not being supported. Marketing a PWA, in terms of it not being in the app store and having to go to it via the website?"

Participant 2: "Yeah I think that. I'm hogging this conversation too much. I think that is an issue. I don't think the world at large have quite clocked this way of doing apps yet. They will, when someone big pushes it. But I think that does knock a little bit off its desirability for some people, they won't see it, they won't even find it."

Participant 3: "Yeah it's a lot different when a TV advert says look for our app on the app store, than visit [www.blahblahblah...](#)"

Participant 1: "There is a way of putting an app container around it though to put it in the app store. Isn't there?"

Participant 2: "There is."

Researcher: "Yeah you could hybrid it."

Participant 2: “You could. Though there’s always been a concern that apple would eventually tell you where to go with that approach. That hasn’t happened as far as I’m aware yet, but it’s always something people have been nervous about that approach.”

Participant 4: “Continuing the conversation from where it was, backing up [participant 2] really, from pure marketing terms there will be a segmentation question, so if you’ve already got essentially a captive audience who you know will follow a URL, for some archaic reason your email marketing campaigns always go really well, a URL is fine. You can market that well. If you’re reliant on the credibility, or the apparent security of using an app store, then for now a wrapper or just making apple behave differently becomes a problem. PWA’s essentially aren’t causing any new questions, it’s just simply a matter of what business you’re in and what your marketing department can tease out for you.”

Participant 3: “It’s also how memorable that URL is, you know if you’ve a particular URL, there’s so many that people do just remember, they’re easy. It’s like it’s not hard to remember BBC is it or something. You know it’s easy.”

Participant 4: “There will be market segments that are cautious about this given certain warnings around only download approved apps from your secure play store. Folks will want to do that, so there will be applications for which that’s not a problem at all; applications with web where the target users are almost to a person going to only go to approved apps.”

Participant 3: “Yeah I think there’s also that next bit with the security, people tend to view a web page more as something they can penetrate than they do an app. They’re going to denial of service, and stuff like that. That stuff is more hit than an app. Because its containerised.”

Participant 2: “Maybe. But...I think there’s another thing that will need to be thought of, not necessarily for you of course, and that’s that whole horrible monetisation of an app. Which adverts, fine, you can drop that into a PWA fairly easily, you want to start taking payments for in-app, you’ve then got to find ways to support that. That users will feel comfortable with. They’ll trust Google and Apple with their money. Cause it goes to them first. You build that yourself or plugin something else, it gets a bit interesting. It will happen, this will happen. It’s not going to be a thing that dies, and there are only native apps. But until someone’s kind of cracked that particular nut, it will be harder to sort of get into your markets with it, a PWA”

Participant 4: “There’s maybe a link back to the question about on device features, you could maybe integrate with Apple pay.”

Participant 2: “Yeah I mean I’m sure those API’s will start. There must be web API’s to do that sort of stuff. I mean you could just put a PayPal if you wanted to...so maybe you just go round it like that. There are just enough little niggles to make people just go, umm, and you don’t need a lot to stop people installing stuff. Security is an interesting one because effectively it’s a webpage right, and people use a browser all the time, but they’ll feel like it’s an app, and not a webpage, but it’s not from an app store. So people will be like, errr... which is not based on any logic. But perception is reality so. That’s again something that I think once...perhaps the answer will be that you need to wrap these and put them through the app store. Maybe that is the right way to do it to make it a consistent experience. And there are loads of apps in the app store that are just web-view... in fact I’m using one for my Facebook because I ripped off my Facebook app because it was killing my phone. So it’s just a web-view onto the browser one effectively. Not that you wanted to know that particularly...”

Participant 3: “But it’s recorded now.”

Participant 2: “Oh yeah, forgot about that.”

Researcher: “Okay, so does anyone have any other comments or is that?”

Participant 2: “I think it looks really cool.”

Participant 3: “Yeah.”

Participant 1: “Yeah does.”

Participant 2: “Really good, I don’t know if we actually said that bit, sorry. Yeah it would be interesting to see, I mean you can only take it so far because you’re doing your project, but it would be interesting to see how far one could take something like that, and try and put that against our app for example. I think the hardest part will be around mapping, although there’s loads of stuff that will do that for you, I don’t think it’s that hard either, but would just be interesting to see. I mean these guys are doing maps all over the place, I mean you could plug your map club map into it and off you go.”

Participant 5: “Yeah.”

Participant 2: “That would be pretty cool. Have you done that already?”

Participant 5: “No.”

Participant 2: “You’re looking like you have. Like yeah I’ve done that...I did that while you were talking about Facebook apps.”

Participant 3: “That’s the only reason he included security at the bottom cause wanted to know how he got into all that stuff.”

Researcher: “Okay that was really good thank you. There was a lot of stuff there I hadn’t really thought about, especially kind of the monetising inside the app.”

Participant 2: “So where are you in terms of what your plans are, are you done from a dev point of view now? You know you’re into report writing? “

Researcher: “Pretty much, I’ve got a couple of weeks left to get all the report ready. I’ve done 5000 words so far, so I’ve got to do another 3000.”

Participant 2: “Easy”

Researcher: “Yeah in terms of, I wanted to put more features in but it’s not really going to get me marks, so I’ve kind of had to stop.”

Participant 2: “That’s very wise, focus on something that proves the point.”

Researcher: “Yeah that’s what I was trying to do, was get something that was, almost an MVP, users could have that and...”

Participant 2: “Yeah that’s a usable product right, at that point already. And yes you could stick other things on it but to be honest, our app went out with not a lot more on it.”

Participant 3: “No didn’t...”

Participant 2: “I mean, other than weather warnings did it have anything else?”


Participant 3: “No that’s right it didn’t have anything”

Participant 2: “I thought it looked very nice, going to have to go play with it now”

Researcher: “The other slide was basically just a, I’ve got an end user survey, you can do it online..., it’s just if you’ve used a weather app before on your phone, so whether you’ve used the Met Office, BBC, whatever you’ve used, to kind of... to load mine up, and then just compare a few things. So compare how it loads, compare how quickly it returns a forecast, and how smooth you rate the animations. It’s a general user survey, if anyone wants to do it on paper know I can give you that, else I’ll email it round. I think with that, that’s pretty much it with my PWA.”

9.8 Appendix 8: Met Office Seminar Presentation Slides

Slide 1

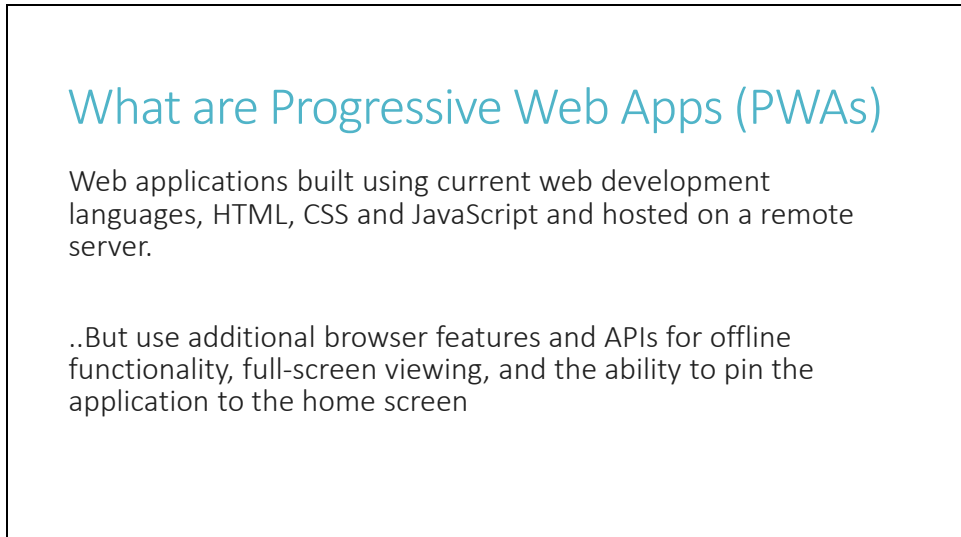
A presentation slide with a solid teal background. The title 'Progressive Web Apps' is written in a large, white, sans-serif font. Below the title, a subtitle in a smaller white font reads: 'Building scalable 'Progressive Web Apps' with comparable performance to that of native mobile applications.' In the bottom right corner, the text 'James Dinan – University of Gloucestershire' is written in a small white font.

Progressive Web Apps

Building scalable 'Progressive Web Apps' with comparable performance to that of native mobile applications.

James Dinan – University of Gloucestershire

Slide 2

A presentation slide with a white background and a thin black border. The title 'What are Progressive Web Apps (PWAs)' is written in a teal, sans-serif font. Below the title, there are two paragraphs of text in a black, sans-serif font. The first paragraph defines PWAs as web applications built using current web development languages, HTML, CSS, and JavaScript, hosted on a remote server. The second paragraph explains that they also use additional browser features and APIs for offline functionality, full-screen viewing, and the ability to pin the application to the home screen.

What are Progressive Web Apps (PWAs)

Web applications built using current web development languages, HTML, CSS and JavaScript and hosted on a remote server.

..But use additional browser features and APIs for offline functionality, full-screen viewing, and the ability to pin the application to the home screen

Slide 3

PWA Requirements

There are 3 elements required for an app to be considered a PWA:

1. **HTTPS**
2. **Web App Manifest** - defines how an application opens, any icons for the home screen, and enables the experience to be "more comparable to that of a native application"
3. **A Service Worker** - script that enables control over how network requests are handled. Allows offline access and push notifications.

Slide 4

...Extras

Progressive enhancement - the adding of more features based on compatibility with new technologies. Chrome, Firefox and Opera browsers for the Android operating system fully compatible.

App Shell - minimal code required for the user interface, for example any navigation. Similar to the UI code packaged within a native app. Any further content can be added progressively later.

Slide 5

Aims of Research Project

1. Develop a weather forecasting PWA and test whether potential optimisation techniques affect the performance of the app.
2. Critique the development complexity and scalability of the application.
3. Gather user feedback on the end experience to see how it compares to native.

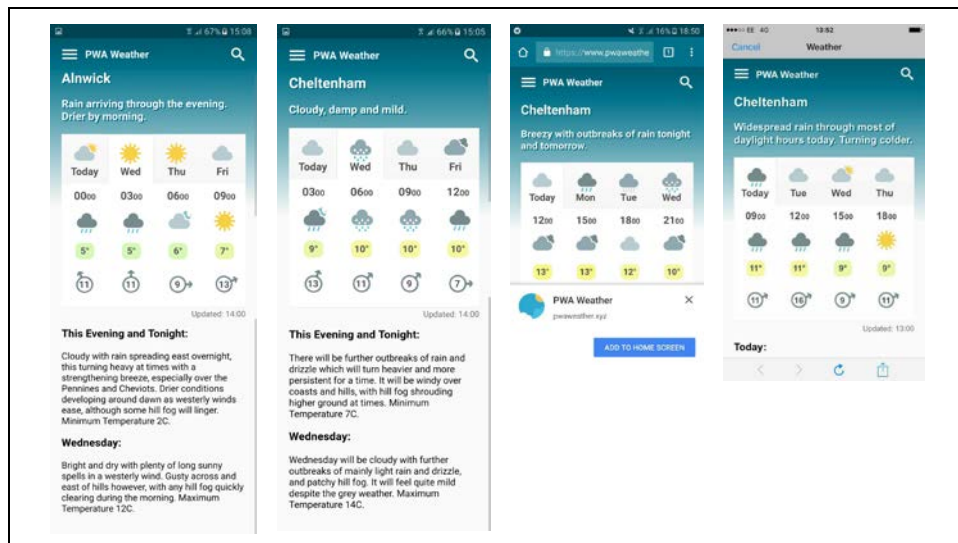
Slide 6

PWA Weather

PWA Weather is a forecast app I've developed that uses the MetOffice DataPoint API. Available at the below link. Add to home screen to enable launching in fullscreen mode.

pwaweather.xyz

Slide 7



Slide 8

PWA benefits compared to native apps

Web applications (in theory) more portable across the different mobile operating systems.

Cheaper development - no need for separate code for multiple platforms

Easily shareable via a URL

Quicker distribution of the latest defect fixes and features.

Slide 9

...but importance of performance

Sites that load within 5 seconds can double the money made through advertising (DoubleClick, 2016, p.12)

49% of individuals expected a native app to perform actions within 2 seconds (Hewlett Packard Enterprise, 2016, p.4) .

48% of individuals have uninstalled a native app that regularly ran slowly, while 33% stopped using the application entirely (Hewlett Packard Enterprise, 2016, p.5)

Slide 10

Some optimisations being tested

1. Non-blocking JS, minification
2. Tree-shaking
3. CSS Transforms, will-change.
4. Web Workers
5. Server-side rendering
6. GraphQL

Slide 11

PWAs instead of native apps?

How would you consider the scalability of PWA Weather?
Would it need any extra functionality that might cause performance problems?

Do you know of any additional or better optimisations that should be considered?

Are JavaScript frameworks (e.g. Angular) or CSS frameworks required or can it be done without?

Slide 12

...PWAs instead of native apps 2

Any on-device hardware/software features not supported by the web?

Would marketing the PWA be viable? Discoverability by users an issue?

How important is security?

Slide 13

User experience survey

There is a user experience survey available below. Requires a smartphone to test PWA Weather. Is targeted at people who have used other weather apps before.

<https://www.esurveycreator.co.uk/s/pwaweather>

Slide 14

Thanks 😊

Thanks very much for coming to my presentation and contributing. Take a treat if you haven't already

9.9 Appendix 9: PWA Weather Code (Dev branch)

The following appendix contains all code created as part of the main development branch of PWA Weather. This defines all application user interface (including the app shell), services for connecting to the Met Office API, Service Worker and Web Manifest files. It additionally includes configuration scripts for minification, tree-shaking, ahead-of-time compilation and creating a release bundle. It also includes the CSS optimisations investigated as part of section 5.4 (Page 9).

Additional branches were created for testing of additional optimisations and for drawing comparisons to existing techniques. The files modified from the dev branch for these are included in Appendices 10 through 15 (Section 12.10 – 12.15).

manifest.json

Manifest file used to declare home-screen icons and display state of application.

```
{
  "short_name": "PWA Weather",
  "name": "PWA Weather",
  "icons": [
    {
      "src": "assets/logo.png",
      "sizes": "96x96",
      "type": "image/png"
    },
    {
      "src": "assets/logo.png",
      "sizes": "144x144",
      "type": "image/png"
    },
    {
      "src": "assets/logo.png",
      "sizes": "192x192",
      "type": "image/png"
    }
  ],
  "start_url": "/index.html",
  "display": "standalone",
  "orientation": "portrait",
  "background_color": "#00728C",
  "theme_color": "#00728C"
}
```

package.json

```
{
  "name": "weather-app",
  "description": "Weather Forecasting Progressive Web Application (PWA).",
  "version": "0.0.1",
  "author": {
    "name": "James Dinan",
    "email": "jamesdinan@connect.glos.ac.uk"
  },
  "repository": {
    "type": "git",
    "url": "https://bitbucket.org/jldinan/weather-app"
  },
  "license": "MIT",
  "scripts": {
    "tsc:watch": "tsc -p tsconfig.json -w",
    "build-release-bundle": "ngc -p tsconfig.aot.json && node rollup.config.js",
    "start": "node server.js"
  },
  "dependencies": {
    "express": "4.15.2",
    "request": "2.81.0"
  },
  "devDependencies": {
    "@angular/common": "2.4.4",
    "@angular/compiler": "2.4.4",
    "@angular/compiler-cli": "2.4.4",
    "@angular/core": "2.4.4",
    "@angular/forms": "2.4.4",
    "@angular/http": "2.4.4",
    "@angular/platform-browser": "2.4.4",
    "@angular/platform-browser-dynamic": "2.4.4",
    "@angular/platform-server": "2.4.4",
    "@angular/router": "3.0.0",
    "@angular/upgrade": "2.0.0",
    "angular2-in-memory-web-api": "0.0.20",
    "browser-sync": "2.18.6",
    "concurrently": "2.2.0",
    "core-js": "2.4.1",
    "gulp": "3.9.1",
    "reflect-metadata": "0.1.9",
    "rollup": "0.41.4",
    "rollup-plugin-commonjs": "7.0.0",
    "rollup-plugin-node-resolve": "2.0.0",
    "rollup-plugin-uglify": "1.0.1",
    "rxjs": "5.0.0-beta.12",
    "systemjs": "0.19.27",
    "systemjs-builder": "0.15.32",
    "@types/node": "6.0.39",
    "typescript": "2.0.3",
    "winston": "2.2.0",
    "zone.js": "^0.6.23"
  },
  "main": "server.js"
}
```

rollup.config.json

```
/**
 * @author James Dinan
 * @date January 2017
 * Tree-Shaking & minification Configuration File.
 * Uses Rollup.js library to tree-shake Angular framework files & application
source code. Compiles into single production bundle to reduce network requests.
 * Tree-shaking removes unused functions within the framework/source code and
reduces bundle size.
 * File minification performed using uglify plugin to further reduce file size.
 */
var rollup = require('rollup'),
    nodeResolve = require('rollup-plugin-node-resolve'),
    commonjs = require('rollup-plugin-commonjs'),
    uglify = require('rollup-plugin-uglify'),
    logger = require('winston');

rollup.rollup({
  entry: './built/main.js',
  plugins: [
    nodeResolve({ module: true }),
    commonjs(),
    uglify()
  ],
  onwarn: function (warning) {
    if (warning.code !== 'THIS_IS_UNDEFINED'){
      logger.log('error', warning.message);
    }
  }
}).then( function ( bundle ) {
  bundle.generate({format: 'iife'});
  bundle.write({
    format: 'iife', dest: './release/app.bundle.min.js', sourceMap: true
  }).then(() => logger.log('info', 'Built Compiled Bundle.'));
});
```

server.js

```
var express = require('express'), app = express(),
    request = require('request'),
    port = (process.env.PORT) ? process.env.PORT : 8080,
    directory = (process.env.PORT) ? '/release' : '';

/**
 * Sets the static directory from which to host the files based on presence of
Amazon port environment variable.
 * Proxy to ensure requests to the Met Office DataPoint API are returned through
server and HTTPS connection.
 */
app.use(express.static(__dirname + directory, { redirect : false }));
app.use('/proxy', function(req, res) {
  var url = req.url.replace('/?url=', '');
  req.pipe(request(url)).pipe(res);
});

app.listen(port, function(){
  console.log('Server running on port: ' + port);
});
exports = module.exports = app;
```

serviceworker.js

```
'use strict';
var cacheVersion = 2,
    currentCache = {offline: 'offline-cache' + cacheVersion},
    assets = [
        '/',
        'index.html',
        'app.bundle.min.js',
        'app-shell.min.js',
        'manifest.json',
        'assets/cancel.png',
        'assets/logo.png',
        'assets/search.png',
        'assets/temperature.png',
        'assets/weather_icon_sprites.png',
        'assets/wind_arrow_sprites.png'
    ];

/**
 * Install the service worker. Adds the assets to the cache.
 */
self.addEventListener('install', function (event) {
    event.waitUntil(caches.open(currentCache.offline).then(function (cache) {
        return cache.addAll(assets).then(function (cache) {
            console.log('assets added');
        });
    }));
});

/**
 * Fetch request.
 * Proxies requests, if no internet connection returns offline assets.
 */
self.addEventListener('fetch', function (event) {
    event.respondWith(caches.match(event.request).then(function (response) {
        return response || fetch(event.request);
    }));
});
```

tsconfig.aot.json

```
{
  "compilerOptions": {
    "module": "es2015",
    "moduleResolution": "node",
    "target": "es5",
    "noImplicitAny": false,
    "sourceMap": false,
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "outDir": "built",
    "declaration": true,
    "lib": ["es2015", "dom"]
  },
  "files": ["app/app.module.ts", "app/main.aot.ts"],
  "angularCompilerOptions": {
    "genDir": "./app/ngfactory",
    "skipMetadataEmit": true
  }
}
```

tsconfig.json

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "system",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false,
    "outFile": "app/bundle.js"
  },
  "types": [
    "core-js",
    "node"
  ],
  "exclude": [
    "gulpfile.js",
    "node_modules",
    "built",
    "app/ngfactory",
    "app/main.aot.ts"
  ]
}
```

gulpfile.js

```
/**
 * @author James Dinan
 * @date January 2017
 * Gulp.js Task Runner Script.
 * Tasks to bundle files, modify script references and minify css. Also used to
 * launch development servers.
 */
var gulp = require('gulp'), browserSync = require('browser-sync'),
    htmlreplace = require('gulp-html-replace'),
    minifycss = require('gulp-minify-css'), babel = require('gulp-babel'),
    uglify = require('gulp-uglify'),
    concat = require('gulp-concat');

//replaces script references in normal develop index.html with references to
//bundled files. minification of app shell css and inserted into head of html file.
gulp.task('index-html', function() {
  gulp.src('./index.html')
    .pipe(htmlreplace({
      'angular': 'app.bundle.min.js',
      'appshelljs': {
        src: 'app-shell.min.js',
        tpl: '<script src="%s" async></script>'
      },
      'appshellcss': {
        src: gulp.src('./app/shell/shell.css').pipe(minifycss()),
        tpl: '<style>%s</style>'
      }
    }))
    .pipe(gulp.dest('release'));
});
```

```

//transpiles es6 javascript for app shell, minifies and concats into single file.
gulp.task('app-shell-javascript', function () {
  return gulp.src(['app/shell/*.js'])
    .pipe(babel({
      presets: ['es2015']
    }))
    .pipe(uglify())
    .pipe(concat('app-shell.min.js'))
    .pipe(gulp.dest('release'));
});

//copies image assets from app directory to release directory.
gulp.task('copy-assets', function () {
  return gulp.src('assets/*')
    .pipe(gulp.dest('release/assets'));
});

//copies manifest file and service worker from app directory to release
directory.
gulp.task('copy-manifest-and-worker', function () {
  return gulp.src(['./manifest.json', './serviceworker.js'])
    .pipe(gulp.dest('release'));
});

//Task to start server hosting release version of application.
gulp.task('serve-release', function () {
  browserSync.init({
    proxy: "localhost:8080",
    ghostMode: false,
    open: 'external'
  });
  gulp.watch('release/app.bundle.min.js').on('change', browserSync.reload);
});

//Task to start server hosting develop version of application.
gulp.task('serve-develop', function () {
  browserSync.init({
    proxy: "localhost:8080",
    ghostMode: false,
    open: 'external'
  });
  gulp.watch('app/bundle.js').on('change', browserSync.reload);
});

//overall build task.
gulp.task('build', ['index-html', 'app-shell-javascript', 'copy-assets', 'copy-
manifest-and-worker']);

```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Weather</title>
  <meta charset="UTF-8">
  <meta name="theme-color" content="#00728C">
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="manifest" href="./manifest.json">
  <!-- Add to home screen for Safari on iOS -->
  <meta name="apple-mobile-web-app-capable" content="yes">
  <meta name="apple-mobile-web-app-status-bar-style" content="black">
  <meta name="apple-mobile-web-app-title" content="PWA Weather">
  <link rel="apple-touch-icon" href="assets/logo.png">
  <!-- build:appshellcss -->
  <link rel="stylesheet" type="text/css" href="app/shell/shell.css">
  <!-- endbuild -->
</head>
<body>
  <header id="header">
    <button class="header-menu"><span></span></button>
    <h1 id="app-title">PWA Weather</h1>
    <search-input id="search-input" class="hidden"></search-input>
    <button class="search-button">
      
    </button>
  </header>
  <aside class="side-nav-wrapper">
    <nav class="side-nav">
      <button class="hide-side-nav-button">CLOSE</button>
      <h2 class="sidenav-header">Previous Locations</h2>
      <div class="content">
        <previous-locations id="previous-locations"></previous-locations>
      </div>
      <h3>Attributions</h3>
      <p class="attributions">Contains public sector information licensed under the Open Government
      Licence.</p>
      <p class="attributions">Weather icons designed by Freepik.</p>
    </nav>
  </aside>
```

```

<div id="app-container">
  <!-- Show a placeholder for content before angular loads-->
  <div id="header-fade-wrapper"></div>
  <div id="search-panel" class="search-panel-hidden">
    <search-locations id="search-locations"></search-locations>
  </div>
  <app></app>
</div>
<!-- build:appshelljs -->
<script src="app/shell/shell.js" async></script>
<!-- endbuild -->
<!-- build:angular -->
<script src="node_modules/core-js/client/shim.min.js"></script>
<script src="node_modules/zone.js/dist/zone.js"></script>
<script src="node_modules/reflect-metadata/Reflect.js"></script>
<script src="node_modules/systemjs/dist/system.src.js"></script>
<script src="app/bundle.js"></script>
<script src="system.config.js"></script>
<script>
  System.import('main').catch(function(err){ console.error(err); });
</script>
<!-- endbuild -->
<script>if('serviceWorker' in navigator) {navigator.serviceWorker.register('./serviceworker.js');}</script>
</body>
</html>

```


app/app.module.ts

```
import {NgModule} from "@angular/core";
import {BrowserModule} from "@angular/platform-browser";
import {HttpModule} from '@angular/http';
import {FormsModule} from "@angular/forms";

import {AppComponent} from "../app.component";
import {HomeComponent} from "../components/home.component";
import {ForecastTableComponent} from "../components/forecast-table/forecast-table.component";
import {PreviousLocationsComponent} from "../components/previous-locations/previous-locations.component";
import {SearchLocationsComponent} from "../components/search-locations/search-locations.component";
import {SearchInputComponent} from "../components/search-input/search-input.component";
import {SelectedLocation} from "../services/selected-location.service";
import {FilterPipe} from "../pipes/search-filter.pipe";

@NgModule({
  imports: [BrowserModule, HttpModule, FormsModule],
  declarations: [
    AppComponent,
    HomeComponent,
    ForecastTableComponent,
    PreviousLocationsComponent,
    SearchLocationsComponent,
    FilterPipe,
    SearchInputComponent
  ],
  providers: [
    SelectedLocation
  ],
  bootstrap: [AppComponent, PreviousLocationsComponent, SearchLocationsComponent, SearchInputComponent]
})
export class AppModule {}
```

app/app.component.ts

```
/**
 * @author James Dinan
 * @date January 2017
 * AppComponent
 * Parent component of the application.
 */
import { Component } from '@angular/core';
@Component({
  selector: 'app',
  template: `<home></home>`
})
export class AppComponent {
  constructor(){}
}
```

app/main.ts

```
import {platformBrowserDynamic} from '@angular/platform-browser-dynamic';
import {AppModule} from './app.module';
import {enableProdMode} from '@angular/core';
enableProdMode();
platformBrowserDynamic().bootstrapModule(<any> AppModule);
```

app/main.aot.ts

```
import 'core-js/client/shim.min.js';
import 'zone.js/dist/zone.js';
import {enableProdMode} from '@angular/core';
import {platformBrowser} from '@angular/platform-browser';
import {AppModuleNgFactory} from './ngfactory/app/app.module.ngfactory';
enableProdMode();
platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);
```

app/shell/shell.js

```
'use strict';
/**
 * @author James Dinan
 * @date January 2017
 * AppShell - transform and will-change version. Minimal JS Code for app-shell navigation and side-nav animation.
 */
class AppShell {
  constructor () {
    //get dom elements.
    this.showButtonEl = document.querySelector('.header-menu');
    this.hideButtonEl = document.querySelector('.hide-side-nav-button');
    this.sideNavEl = document.querySelector('.side-nav-wrapper');
    this.sideNavContainerEl = document.querySelector('.side-nav');
    this.searchPanel = document.querySelector('#search-panel');
    this.searchButton = document.querySelector('.search-button');
    this.searchButtonIcon = document.querySelector('.search-button-icon');
    this.appTitle = document.querySelector('#app-title');
    this.searchInput = document.querySelector('#search-input');
    this.searchLocations = document.querySelector('#search-locations');
    this.previousLocations = document.querySelector('#previous-locations');
    this.searchInputBox;

    //Bind to events.
    this.showSearchPanel = this.showSearchPanel.bind(this);
    this.showSideNav = this.showSideNav.bind(this);
    this.hideSideNav = this.hideSideNav.bind(this);
    this.blockClicks = this.blockClicks.bind(this);
    this.onTouchStart = this.onTouchStart.bind(this);
    this.onTouchMove = this.onTouchMove.bind(this);
    this.onTouchEnd = this.onTouchEnd.bind(this);
    this.onTransitionEnd = this.onTransitionEnd.bind(this);
    this.update = this.update.bind(this);

    //init variables and add event listeners
    this.startX = 0;
    this.currentX = 0;
    this.touchingSideNav = false;
    this.supportsPassive = undefined;
    this.showingSearch = false;
    this.addEventListener();
  }
}
```

```

/**
 * Adds event listeners to button and container elements.
 */
addEventListeners () {
  var global = this;
  this.searchButton.addEventListener('click', this.showSearchPanel);
  this.searchLocations.addEventListener('click', this.showSideNav);
  this.previousLocations.addEventListener('click', function(){global.hideSideNav(200);});
  this.showButtonEl.addEventListener('click', this.showSideNav);
  this.hideButtonEl.addEventListener('click', this.hideSideNav);
  this.sideNavEl.addEventListener('click', this.hideSideNav);
  this.sideNavContainerEl.addEventListener('click', this.blockClicks);
  this.sideNavEl.addEventListener('touchstart', this.onTouchStart, this.applyPassive());
  this.sideNavEl.addEventListener('touchmove', this.onTouchMove, this.applyPassive());
  this.sideNavEl.addEventListener('touchend', this.onTouchEnd);
}

showSearchPanel () {
  if (this.showingSearch){
    if (!this.searchInputBox){
      this.searchInputBox = document.querySelector('.search-input');
      this.searchInputBox.value = '';
      this.searchInputBox.focus();
    } else {
      this.searchInputBox.value = '';
      this.searchInputBox.focus();
    }
  } else {
    this.showingSearch = true;
    this.searchPanel.classList.remove('search-panel-hidden');
    this.searchPanel.classList.add('search-panel-show');
    this.appTitle.classList.add('hidden');
    this.appTitle.classList.remove('show');
    this.searchInput.classList.add('show');
    this.searchInput.classList.remove('hidden');
    this.searchButtonIcon.src="assets/cancel.png";
    this.showButtonEl.classList.add('is-active');
    if (!this.searchInputBox){
      this.searchInputBox = document.querySelector('.search-input');
      this.searchInputBox.focus();
    } else {
      this.searchInputBox.focus();
    }
  }
}

```

```

    }
  }
}

/**
 * On Touch Start event, manages dragging of side-nav.
 * Calls requestAnimationFrame() with current x position of touch event.
 * @param {Event} evt
 */
onTouchStart (evt) {
  if (!this.sideNavEl.classList.contains('side-nav--visible')){return;}
  this.startX = evt.touches[0].pageX;
  this.currentX = this.startX;
  this.touchingSideNav = true;
  requestAnimationFrame(this.update);
}

/**
 * Sets the current x position of side-nav on touch move event.
 * @param {Event} evt
 */
onTouchMove (evt) {
  if (!this.touchingSideNav) {return;}
  this.currentX = evt.touches[0].pageX;
}

/**
 * Touch End event, hides the side-nav when user lets go of drag.
 * Point at which side-nav returns to hidden state controlled by less than check against x position.
 */
onTouchEnd () {
  if (!this.touchingSideNav){return;}
  this.touchingSideNav = false;
  const translateX = Math.min(0, this.currentX - this.startX);
  this.sideNavContainerEl.style.transform = '';
  if (translateX < 0) {
    this.hideSideNav(0);
  }
}
}

```

```

/**
 * Sets the position of the side-nav based on the current and starting x coordinate.
 */
update () {
  if (!this.touchingSideNav) {return;}
  requestAnimationFrame(this.update);
  const translateX = Math.min(0, this.currentX - this.startX);
  this.sideNavContainerEl.style.transform = `translateX(${translateX}px)`;
}

/**
 * Blocks clicks on side-nav container.
 * @param {Event} evt
 */
blockClicks (evt) {
  evt.stopPropagation();
}

/**
 * On ending transition, removes animation class and event listener.
 */
onTransitionEnd () {
  this.sideNavEl.classList.remove('side-nav--animatable');
  this.sideNavEl.removeEventListener('transitionend', this.onTransitionEnd);
}

/**
 * Adds animation and visible classes to side-nav.
 * Allows it to animate into view.
 */
showSideNav () {
  if (this.showingSearch){
    this.showingSearch = false;
    this.searchPanel.classList.add('search-panel-hidden');
    this.searchPanel.classList.remove('search-panel-show');
    this.appTitle.classList.add('show');
    this.appTitle.classList.remove('hidden');
    this.searchInput.classList.add('hidden');
    this.searchInput.classList.remove('show');
    this.showButtonEl.classList.remove('is-active');
    this.searchButtonIcon.src="assets/search.png";
  } else {

```

```

        this.sideNavEl.classList.add('side-nav--animatable');
        this.sideNavEl.classList.add('side-nav--visible');
        this.sideNavEl.addEventListener('transitionend', this.onTransitionEnd);
    }
}

/**
 * Removes the visible class causing side-nav to transition out of view.
 */
hideSideNav (delay) {
    var global = this;
    if (delay) {
        setTimeout(function(){
            global.sideNavEl.classList.add('side-nav--animatable');
            global.sideNavEl.classList.remove('side-nav--visible');
            global.sideNavEl.addEventListener('transitionend', global.onTransitionEnd);
        }, delay);
    } else {
        global.sideNavEl.classList.add('side-nav--animatable');
        global.sideNavEl.classList.remove('side-nav--visible');
        global.sideNavEl.addEventListener('transitionend', global.onTransitionEnd);
    }
}

/**
 * Checks whether current browser supports passive adding of event listeners.
 * Passive support limits blocking of UI thread.
 * @returns {*}
 */
applyPassive () {
    let isSupported = false;
    if (this.supportsPassive !== undefined) {return this.supportsPassive ? {passive: true} : false;}
    try {
        document.addEventListener('test', null, {get passive () {isSupported = true;}});
    } catch (e) { }
    this.supportsPassive = isSupported;
    return this.applyPassive();
}
}
new AppShell();

```

app/shell/shell.css

```
/**
 * @author James Dinan
 * @date February 2017 *
 * Minimal CSS styling for app-shell, navigation and side-nav animation.
 */
* {
  box-sizing: border-box;
}
html, body {
  padding: 0;
  margin: 0;
  background: #FAFAFA;
  font-family: Arial, sans-serif;
  overflow: hidden;
  height: 100%;
}
button {
  outline: none;
}
#app-container {
  padding: 5px 20px 20px 20px;
  overflow-y: auto;
  overflow-x: hidden;
  height: calc(100% - 56px);
}
#header {
  align-items: center;
  background: #00728C;
  color: #FFF;
  display: flex;
  flex-direction: row;
  height: 56px;
  padding: 0 16px;
  width: 100%;
  background: linear-gradient(#005b70, #00728C);
}
#header h1 {
  color: rgb(242, 248, 255);
  font-size: 18px;
  margin-left: 10px;
  margin-top: 16px;
}
#header .header-menu {
  position: relative;
  margin: 0;
  padding: 0;
  width: 35px;
  height: 35px;
  font-size: 0;
  background: none !important;
  border: none;
  box-shadow: none;
  border-radius: 10px;
}
#header .header-menu span {
  display: block;
  position: absolute;
  top: 16px;
```



```

    left: 5px;
    right: 4px;
    height: 3px;
    background: white;
    transition: transform 0.3s;
}
#header .header-menu span::before, .header-menu span::after {
    position: absolute;
    display: block;
    left: 0;
    width: 100%;
    height: 3px;
    background-color: #fff;
    content: "";
    border-radius: 64px;
}
#header .header-menu span::before {
    top: -7px;
}
#header .header-menu span::after {
    bottom: -7px;
}
#header .header-menu span::before {
    transform-origin: top right;
    transition: transform 0s, width 0s, top 0s;
}
#header .header-menu span::after {
    transform-origin: bottom right;
    transition: transform 0s, width 0s, bottom 0s;
}
#header .header-menu.is-active span {
    transform: rotate(180deg);
    display: block;
    position: absolute;
    top: 16px;
    left: 8px;
    right: 4px;
    height: 3px;
    background: white;
    width: 20px;
}
#header .header-menu.is-active span::before,
#header .header-menu.is-active span::after {
    width: 60%;
    left: -20px;
}
#header .header-menu.is-active span::before {
    top: 0;
    transform: translateX(29px) translateY(1px) rotate(45deg);
}
#header .header-menu.is-active span::after {
    bottom: 0;
    transform: translateX(29px) translateY(-1px) rotate(-45deg);
}
#header .search-button {
    border-radius: 10px;
    width: 35px;
    height: 35px;
    font-size: 0;
    background: none !important;
}

```

```

    border: none;
    box-shadow: none;
    margin-right: 0;
    margin-left: auto;
}
#header .search-button img {
    width: 23px;
}
#header button:active {
    background: #00728c !important;
}
#header-fade-wrapper {
    height: 250px;
    margin-left: -20px;
    margin-top: -5px;
    width: calc(100% + 40px);
    background-color: #00728C;
    background: linear-gradient(#00728C, #fafafa);
}
#search-panel {
    width: 100%;
    height: 100%;
    position: absolute;
    z-index: 30;
    top: 56px;
    left: 0;
    background: #FAFAFA;
    will-change: opacity;
    overflow: auto;
}
#search-input {
    background: none;
    border: none;
    color: white;
    outline: none;
    font-size: 18px;
    font-weight: bold;
    height: 30px;
    width: calc(100% - 85px);
    text-decoration-color: white;
}
.search-panel-show {
    opacity: 1;
    transition: opacity 0.3s cubic-bezier(0,0,0.3,1);
    pointer-events: auto;
}
.search-panel-hidden {
    opacity: 0;
    transition: opacity 0.6s cubic-bezier(0,0,0.3,1) 0.4s;
    pointer-events: none;
}
.hidden {
    display: none;
}
.show {
    display: block;
}
.side-nav-wrapper {
    position: fixed;
    left: 0;
    top: 0;

```

```

    width: 100%;
    height: 100%;
    overflow: hidden;
    pointer-events: none;
}
.side-nav-wrapper::before {
    content: '';
    display: block;
    position: absolute;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    background: rgba(0,0,0,0.4);
    opacity: 0;
    will-change: opacity;
    transition: opacity 0.3s cubic-bezier(0,0,0.3,1);
}
.side-nav {
    background-color: white;
    display: flex;
    flex-direction: column;
    height: 100%;
    max-width: 400px;
    position: relative;
    transform: translateX(-102%);
    will-change: transform;
    width: 90%;
    padding: 20px;
}
.side-nav .sidenav-header {
    margin-bottom: 5px;
    margin-top: 20px;
}
.side-nav .hide-side-nav-button {
    position: absolute;
    right: 12px;
    top: -3px;
    background: none;
    border: none;
    color: black;
    width: 50px;
    height: 50px;
    padding: 0;
    margin: 0;
}
.side-nav .content {
    flex: 1;
    list-style: none;
    padding: 0;
    margin: 0;
    overflow-x: hidden;
    overflow-y: auto;
    -webkit-overflow-scrolling: touch;
    width: calc(100% + 12px);
    margin-left: -6px;
}
.side-nav h3 {
    margin-bottom: 0;
    color: #505050;
}

```

```
.side-nav .attributions {
  font-size: 14px;
  margin-top: 7px;
  margin-bottom: 4px;
  color: #505050;
}
.side-nav--visible {
  pointer-events: auto;
}
.side-nav--animatable .side-nav {
  transition: transform 0.2s cubic-bezier(0,0,0.3,1);
}
.side-nav--visible.side-nav--animatable .side-nav {
  transition: transform 0.2s cubic-bezier(0,0,0.3,1);
}
.side-nav--visible::before {
  opacity: 1;
}
.side-nav--visible .side-nav {
  transform: none;
}
```

app/services/forecast.service.ts

```
/**
 * @author James Dinan
 * @date January 2017
 * Forecast Service.
 * Pulls available 3-Hourly site specific forecasts and text-based forecasts from the Met Office DataPoint API.
 */
import {Injectable} from "@angular/core";
import {Http, Response} from "@angular/http";
import {Observable} from 'rxjs/Observable';
import 'rxjs/add/operator/catch';
import 'rxjs/add/operator/map';
import 'rxjs/add/observable/throw';

@Injectable()
export class ForecastService {
  constructor (private http: Http) {}

  /**
   * Get request to retrieve the 3-Hourly forecast data from the Met Office API for a specific siteID.
   * @param {Array} dailyForecast
   * @param {string} siteID
   * @returns {Observable<R>}
   */
  get3HourlyForecast(dailyForecast, siteID: string) {
    return this.http.get('/proxy?url=http://datapoint.metoffice.gov.uk/public/data/val/wxfcs/all/json/' + siteID
      + '?res=3hourly&key=')
      .map((res) => this.extractData(dailyForecast, res))
      .catch((error) => this.handleError());
  }

  /**
   * Get request to retrieve the daily forecast data from the Met Office API for a specific siteID.
   * @param {string} siteID
   * @returns {Observable<R>}
   */
  getDailyForecastData(siteID: string){
    return this.http.get('/proxy?url=http://datapoint.metoffice.gov.uk/public/data/val/wxfcs/all/json/' + siteID
      + '?res=daily&key=')
      .map((res) => this.extractDailyForecastData(res))
      .catch((error) => this.handleError());
  }
}
```

```

/**
 * Get request to retrieve the text-based forecast from the Met Office API for a specific siteID.
 * @param {string} siteID
 * @returns {Observable<R>}
 */
getTextForecastData(siteID: string){
    return this.http.get('/proxy?url=http://datapoint.metoffice.gov.uk/public/data/txt/wxfcs/
        regionalforecast/json/' + siteID + '?key=')
        .map((res) => this.extractTextData(res))
        .catch((error) => this.handleError());
}

/**
 * Extracts the 3-hourly forecast data.
 * @param dailyForecast
 * @param {Response} res
 * @returns {Array}
 */
private extractData(dailyForecast, res: Response) {
    let forecastData = res.json().SiteRep.DV.Location.Period,
        forecast = [],
        updated = new Date(res.json().SiteRep.DV.dataDate);
    if (forecastData instanceof Array){
        for (let index in forecastData) {
            let day = forecastData[index],
                timesteps = this.extractTimesteps(day),
                dayWeatherType = (dailyForecast[day.value].weatherType)?dailyForecast[day.value].weatherType: 7;
            forecast.push({
                updated: updated.getHours() + ":" + (updated.getMinutes() < 10 ? '0' : '') + updated.getMinutes(),
                date: day.value,
                day: this.getFormattedDay(new Date(day.value.replace(/-/g, "/").replace(/Z/g, " ")).getDay()),
                weatherType: dayWeatherType,
                weatherTypeIconClass: this.getWeatherTypeIconClass(dayWeatherType),
                timesteps: timesteps
            });
        }
    }
    return forecast;
}

```

```

/**
 * Extracts the daily forecast data.
 * @param {Response} res
 * @returns {Array}
 */
private extractDailyForecastData(res: Response) {
  let forecastData = res.json().SiteRep.DV.Location.Period,
      forecast = [];
  if (forecastData instanceof Array){
    for (let index in forecastData) {
      let day = forecastData[index];
      forecast[day.value] = {
        date: day.value,
        weatherType: day.Rep[0].W
      };
    }
  }
  return forecast;
}

/**
 * Extracts the text-based forecast data.
 * @param {Response} res
 * @returns {{headline: any, timestep1: {title: any, summary: any}, timestep2: {title: any, summary: any}}}
 */
private extractTextData(res: Response) {
  let forecastTextData = res.json().RegionalFcst.FcstPeriods.Period;
  return {
    headline: forecastTextData[0].Paragraph[0]["$"],
    timestep1: {
      title: forecastTextData[0].Paragraph[1]["title"],
      summary: forecastTextData[0].Paragraph[1]["$"]
    },
    timestep2: {
      title: forecastTextData[0].Paragraph[2]["title"],
      summary: forecastTextData[0].Paragraph[2]["$"]
    }
  };
}

```

```

/**
 * Extracts and processes the timestep data from a given day object.
 * @param {object} day
 * @returns {{temperature: number}[]}
 */
private extractTimesteps(day: any): any{
    let daySteps = day.Rep,
        timesteps = [];
    if (daySteps instanceof Array){
        for (let index in daySteps) {
            let timestep = daySteps[index];
            timesteps.push(
                {
                    weatherType: timestep.W,
                    weatherTypeIconClass: this.getWeatherTypeIconClass(timestep.W),
                    windDirection: timestep.D,
                    windSpeed: timestep.S,
                    windGust: timestep.G,
                    temperature: timestep.T,
                    temperatureClass: this.getTemperatureColourClass(timestep.T),
                    feelsLikeTemperature: timestep.F,
                    humidity: timestep.H,
                    precipProbability: timestep.Pp,
                    visibility: timestep.V,
                    uv: timestep.U,
                    time: this.getFormattedTime(timestep.$ / 60)
                }
            );
        }
    }
    return timesteps;
}

/**
 * Handles error event from the API.
 * @returns {ErrorObservable}
 */
private handleError () {
    let errMsg: string;
    errMsg = 'Error retrieving forecast data.';
    return Observable.throw(errMsg);
}

```



```

/**
 * Returns the day string for a given day index number.
 * @param {number} day
 * @returns {string|string}
 */
private getFormattedDay(day){
    let weekday = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"];
    return (new Date().getDay() === day) ? "Today" : weekday[day];
}

/**
 * Returns the time param, hours since midnight, in 24 hour format.
 * @param {number} time
 * @returns {string}
 */
private getFormattedTime(time){
    return (time < 10) ? '0' + time : '' + time;
}

/**
 * Returns the temperature class corresponding to the given temperature param.
 * @param {number} temperature
 * @returns {string}
 */
private getTemperatureColourClass(temperature){
    let tempClass = 'temp-0';
    if (temperature > 0 && temperature <= 3) {tempClass = 'temp-1-3';}
    else if (temperature > 3 && temperature <= 6) {tempClass = 'temp-4-6';}
    else if (temperature > 6 && temperature <= 9) {tempClass = 'temp-7-9';}
    else if (temperature > 9 && temperature <= 12) {tempClass = 'temp-10-12';}
    else if (temperature > 12 && temperature <= 15) {tempClass = 'temp-13-15';}
    else if (temperature > 15 && temperature <= 18) {tempClass = 'temp-16-18';}
    else if (temperature > 18 && temperature <= 21) {tempClass = 'temp-19-21';}
    else if (temperature > 21 && temperature <= 24) {tempClass = 'temp-22-24';}
    else if (temperature > 24) {tempClass = 'temp-25';}
    return tempClass;
}

```

```

/**
 * Returns the weather icon class for the given weather type param.
 * @param {number} type
 * @returns {string|string}
 */
private getWeatherTypeIconClass(type){
    let types = [
        "clear-night",
        "sunny-day",
        "partly-cloudy-night",
        "partly-cloudy-day",
        "",
        "mist",
        "fog",
        "cloudy",
        "overcast",
        "light-rain-shower-night",
        "light-rain-shower-day",
        "drizzle",
        "light-rain",
        "heavy-rain-shower-night",
        "heavy-rain-shower-day",
        "heavy-rain",
        "sleet-shower-night",
        "sleet-shower-day",
        "sleet",
        "hail-shower-night",
        "hail-shower-day",
        "hail",
        "light-snow-shower-night",
        "light-snow-shower-day",
        "light-snow",
        "heavy-snow-shower-night",
        "heavy-snow-shower-day",
        "heavy-snow",
        "thunder-shower-night",
        "thunder-shower-day",
        "thunder"
    ];
    return (isNaN(type)) ? '' : types[type];
}

```

app/services/location.service.ts

```
/**
 * @author James Dinan
 * @date January 2017
 * Location Service.
 * Pulls available forecast locations from the Met Office DataPoint API.
 */
import {Injectable} from "@angular/core";
import {Http, Response} from "@angular/http";
import {Observable} from 'rxjs/Observable';
import 'rxjs/add/operator/catch';
import 'rxjs/add/operator/map';
import 'rxjs/add/observable/throw';
import {ForecastLocation} from "../services/location";

@Injectable()
export class LocationService {
  constructor (private http: Http) {}

  /**
   * Get request to retrieve the forecast locations from the Met Office API
   * @returns {Observable<R>}
   */
  getLocations(): Observable<ForecastLocation[]> {
    return this.http.get('/proxy?url=http://datapoint.metoffice.gov.uk/public/data/val/wxfcs  

    /all/json/sitelist?key=')
    .map((res) => this.extractData(res))
    .catch((error) => this.handleError());
  }

  /**
   * Extracts the data and sorts alphabetically.
   * @param {Response} res
   * @returns {Array}
   */
  private extractData(res: Response) {
    var locationData = res.json().Locations.Location,
        locations = [];
    for (let location in locationData) {
      var loc = locationData[location];
      locations.push({

```

```

        id:loc.id,
        name:loc.name,
        lat:loc.latitude,
        lon:loc.longitude,
        region: loc.region,
        regionID: this.getRegionalID(loc.region)
    });
}
locations.sort(function(a, b) {
    return a.name.localeCompare(b.name);});
return locations;
}

/**
 * Handles error event from the API.
 * @returns {ErrorObservable}
 */
private handleError() {
    let errMsg: string;
    errMsg = 'Error retrieving forecast locations.';
    return Observable.throw(errMsg);
}

/**
 * Returns the regionalID for a given region name string.
 * @param {string} region
 * @returns {any}
 */
private getRegionalID(region){
    let regions = [
        {"@id": "500", "@name": "os"}, {"@id": "501", "@name": "he"}, {"@id": "502", "@name": "gr"},
        {"@id": "503", "@name": "st"}, {"@id": "504", "@name": "ta"}, {"@id": "505", "@name": "dg"},
        {"@id": "506", "@name": "ni"}, {"@id": "507", "@name": "nw"}, {"@id": "508", "@name": "ne"},
        {"@id": "509", "@name": "yh"}, {"@id": "510", "@name": "wm"}, {"@id": "511", "@name": "em"},
        {"@id": "512", "@name": "ee"}, {"@id": "513", "@name": "sw"}, {"@id": "514", "@name": "se"},
        {"@id": "515", "@name": "uk"}, {"@id": "516", "@name": "wl"}];
    for(var i = 0, len = regions.length; i < len; i++) {
        if (regions[i]["@name"] === region){return regions[i]["@id"];}
    }
    return "515";
}
}

```

app/services/selected-location.service.ts

```
/**
 * @author James Dinan
 * @date March 2017
 * SelectedLocation Service.
 * Allows the selected forecast location to be shareable amongst components outside of the root application.
 */
import {Injectable, EventEmitter} from "@angular/core";
@Injectable()
export class SelectedLocation {
  public selectedLocation$: EventEmitter<any>;
  public searchLocationTerm$: EventEmitter<any>;
  constructor () {
    this.selectedLocation$ = new EventEmitter();
    this.searchLocationTerm$ = new EventEmitter();
  }

  /**
   * Used to set the selectedLocation. Emits event watched by components.
   * @param {object} location
   */
  public setLocation(location): void {
    this.selectedLocation$.emit(location);
  }

  /**
   * Used to set the search location term. Emits event watched by components.
   * @param {string} term
   */
  public setSearchLocationTerm(term): void {
    this.searchLocationTerm$.emit(term);
  }
}
```

app/services/location.ts

```
export interface ForecastLocation {
  id: number, name: string, lat: number, lon: number, region: string
}
```

app/components/home.component.ts

```
/**
 * @author James Dinan
 * @date January 2017
 * HomeComponent - Main child component of the application.
 * Loads forecast data, styles main content and includes forecast-table component.
 */
import {Component} from "@angular/core";
import {ForecastService} from "../../services/forecast.service";
import {SelectedLocation} from "../../services/selected-location.service";

@Component({
  selector: 'home',
  template: `
    <div id="headline-container">
      <h2>{{selectedLocation.name}}</h2>
      <h3>{{forecastText?.headline}}</h3>
    </div>
    <forecast-table [forecast]="forecast" [selectedForecast]="selectedForecast"></forecast-table>
    <div [ngClass]="{'display': forecast.length > 0}" id="forecastText">
      <h3>{{forecastText?.timestep1?.title}}</h3>
      <p>{{forecastText?.timestep1?.summary}}</p>
      <h3>{{forecastText?.timestep2?.title}}</h3>
      <p>{{forecastText?.timestep2?.summary}}</p>
    </div>
    <style>
      #headline-container {
        min-height: 90px;
        color: white;
        text-shadow: 2px 2px #00677f;
        margin-top: -259px;
        padding-bottom: 20px;
      }
      #headline-container h3 {margin-bottom: 0;}
      #forecastText {margin-top: 35px; display: none;}
      .display {display: block !important;}
    </style>
  `,
  providers: [ForecastService]
})
```

```

export class HomeComponent {
  public forecast = [];
  public forecastError:String = '';
  public selectedForecast = [];
  public selectedLocation = {"id": "310047", "name": "Cheltenham",
    "lat": "51.8972", "lon": "-2.0742", "region": "sw", "regionID": "513"};
  public forecastText = {headline: "", timestep1: {title: "", summary: ""}, timestep2: {title: "", summary: ""}};

  constructor(private ForecastService:ForecastService,
    selectedLocation: SelectedLocation
  ) {
    selectedLocation.selectedLocation$.subscribe(location => this.onLocationChanged(location));
  }

  /**
   * Initialises HomeComponent - makes service calls to retrieve locations,
   * forecast tabular data and regional textual forecast.
   */
  ngOnInit():any {
    this.getTabularForecastData();
    this.getTextForecast();
  }

  /**
   * Triggered when selectedLocation is changed within the shared service.
   * Makes call to service functions to retrieve new forecast data.
   * @param {object} location
   */
  private onLocationChanged(location): void {
    this.selectedLocation = location;
    this.getTabularForecastData();
    this.getTextForecast();
  }

  /**
   * Calls service functions to retrieve the daily forecast data,
   * and then 3-Hourly Forecast data from Met Office DataPoint API for the selectedLocation.
   * Sets class variables and error state accordingly.
   */
  getTabularForecastData(){
    this.ForecastService.getDailyForecastData(this.selectedLocation.id).subscribe(
      res => {

```

```

        this.ForecastService.get3HourlyForecast(res, this.selectedLocation.id).subscribe(
            res => {
                this.forecast = res;
                this.selectedForecast = this.forecast[0];
            },
            error => this.forecastError = <any>error
        );
    },
    error => this.forecastError = <any>error
);
}

/**
 * Calls service function to retrieve regional text data from Met Office DataPoint API for the selectedLocation.
 * Sets class variables and error state accordingly.
 */
getTextForecast(){
    this.ForecastService.getTextForecastData(this.selectedLocation.regionID).subscribe(
        res => {
            this.forecastText = res;
        },
        error => this.forecastError = <any>error
    );
}
}

```


app/components/forecast-table.component.ts

```
/**
 * @author James Dinan
 * @date January 2017
 * ForecastTableComponent
 * Renders the forecast table using the defined input params.
 */
import {Component, Input} from "@angular/core";

@Component({
  moduleId: 'app/components/forecast-table/',
  selector: 'forecast-table',
  styleUrls: ['forecast-table.css'],
  template: `
    <div id="forecast-table-container">
      <ul>
        <li [ngClass]="{'active-tab': i === selectedIndex}" (click)="changeForecastDay(i)" *ngFor="let day of
          forecast; let i = index">
          <div [ngClass]="day.weatherTypeIconClass" class="weather-type"></div>
          <span>{{day.day}}</span>
        </li>
      </ul>
      <div [ngClass]="{'loaded-table': forecast.length > 0}" class="table-container">
        <table>
          <thead>
            <tr>
              <th class="timestep" *ngFor="let timestep of selectedForecast.timesteps">
                <span class="hour">{{timestep.time}}</span>
                <span class="minutes">00</span>
              </th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td *ngFor="let timestep of selectedForecast.timesteps">
                <div [ngClass]="timestep.weatherTypeIconClass" class="weather-type"></div>
              </td>
            </tr>
            <tr>
              <td *ngFor="let timestep of selectedForecast.timesteps">
                <div [ngClass]="timestep.temperatureClass"
```

```

        class="temperature">{{timestep.temperature}}&deg;</div>
      </td>
    </tr>
  <tr>
    <td *ngFor="let timestep of selectedForecast.timesteps">
      <div [ngClass]="timestep.windDirection" class="wind-arrow">
        <span class="wind-speed">{{timestep.windSpeed}}</span>
      </div>
    </td>
  </tr>
</tbody>
</table>
</div>
<span [ngClass]="{'display': forecast.length > 0}" class="updated">Updated: {{forecast[0]?.updated}} </span>
</div>`,
providers: []
})
export class ForecastTableComponent {
  @Input() forecast;
  @Input() selectedForecast;
  public selectedIndex = 0;
  constructor() {}

  /**
   * Resets the selected table index when any of the forecast data changes.
   */
  ngOnChanges(changes: any) {
    this.selectedIndex = 0;
  }

  /**
   * Changes the selectedForecast to the forecast array matching the passed in index.
   * @param {number} index
   */
  changeForecastDay(index){
    this.selectedForecast = this.forecast[index];
    this.selectedIndex = index;
  }
}

```

app/components/forecast-table/forecast-table.css

```
/*
 * @author James Dinan
 * @date February 2017
 * Styling for the ForecastTableComponent.
 */
ul {
  width: 100%;
  padding: 0;
  overflow-x: auto;
  margin: 0;
  white-space: nowrap;
  background-color: #f7f7f7;
  background: linear-gradient(#FAFAFA, #f7f7f7);
}
ul li {
  font-size: 18px;
  color: #505050;
  display: inline-block;
  width: 80px;
  text-align: center;
  padding: 10px;
  font-weight: bold;
}
ul li:first-child {
  border-left: none !important;
}
ul li:last-child {
  border-right: none !important;
}
table {
  border-collapse: collapse;
  display: block;
  max-width: 100%;
  overflow-y: auto;
  color: #505050;
  background-color: white;
}
td, th {
  padding: 0.5rem;
  text-align: center;
  border-top: none;
  border-bottom: none;
}
.table-container {
  width: 100%;
  box-sizing: border-box;
  margin-bottom: 4px;
}
.loaded-table {
  border-bottom: solid 1px #eaeaea;
}
.active-tab {
  border: solid 1px #eaeaea;
  background-color: white;
  border-bottom: none;
}
.weather-type {
  width: 50px;
```

```

    height: 50px;
    background: url(../../assets/weather_icon_sprites.png);
    background-size: 600% 600%;
    margin: 0 auto -5px auto;
    background-position: 947px -11px;
}
.timestep {
    min-width: 80px;
    text-align: center;
    padding: 10px 0 2px 0;
}
.hour {
    font-size: 19px;
    color: #505050;
}
.minutes {
    font-size: 14px;
    color: #505050;
    letter-spacing: 0;
    margin-left: -4px;
}
.temperature {
    min-width: 33px;
    background-color: lightblue;
    padding: 5px;
    display: inline-block;
    text-align: center;
    border-radius: 10px;
    font-weight: bold;
}
.wind-arrow {
    width: 65px;
    height: 65px;
    background: url(../../assets/wind_arrow_sprites.png);
    background-size: 450% 450%;
    margin: 0 auto -5px auto;
    background-position: 0 -18px;
}
.wind-speed {
    font-weight: bold;
    margin-top: 0;
    margin-left: 22px;
    text-align: center;
    width: 19px;
    display: block;
    padding-top: 19px;
}
.updated {
    float: right;
    display: none;
    color: #717171;
    font-size: 14px;
}
.display {
    display: block;
}

/*Temperature colour values*/
.temp-0 {

```

```

        background-color: lightblue;
    }
    .temp-1-3 {
        background-color: #bcf9d4;
    }
    .temp-4-6 {
        background-color: #d7f9b8;
    }
    .temp-7-9 {
        background-color: #eaf9ab;
    }
    .temp-10-12 {
        background-color: #f8f9a2;
    }
    .temp-13-15 {
        background-color: #f9f988;
    }
    .temp-16-18 {
        background-color: #fff979;
    }
    .temp-19-21 {
        background-color: #ffe960;
    }
    .temp-22-24 {
        background-color: #ffda48;
    }
    .temp-25 {
        background-color: #ffcc54;
    }
}

/*Weather type icon classes.*/
.overcast {
    background-position: 883px -20px;
}
.sunny-day {
    background-position: 828px -16px;
}
.cloudy {
    background-position: 777px -20px;
}
.clear-night {
    background-position: 669px -20px;
}
.sleet-shower-day {
    background-position: 723px -20px;
}
.sleet {
    background-position: 723px -20px;
}
.hail-shower-day {
    background-position: 723px -20px;
}
.hail {
    background-position: 723px -20px;
}
.thunder {
    background-position: 968px -70px;
}

.sleet-shower-night {
    background-position: 1020px -70px;
}

```

```

}
.hail-shower-night {
    background-position: 1020px -70px;
}
.heavy-rain-shower-day {
    background-position: 1080px -70px;
}
.heavy-rain {
    background-position: 1080px -70px;
}
.light-snow-shower-day {
    background-position: 1131px -70px;
}
.light-snow {
    background-position: 1131px -70px;
}
.partly-cloudy-day {
    background-position: 1184px -74px;
}
.partly-cloudy-night {
    background-position: 1184px -129px;
}
.light-snow-shower-night {
    background-position: 1129px -125px;
}
.heavy-rain-shower-night {
    background-position: 1076px -125px;
}
.heavy-snow-shower-day {
    background-position: 1021px -124px;
}
.heavy-snow {
    background-position: 1021px -124px;
}
.sunrise {
    background-position: 967px -124px;
}
.fog {
    background-position: 967px -173px;
}
.mist {
    background-position: 1020px -176px;
}
.thunder-shower-night {
    background-position: 1076px -176px;
}
.thunder-shower-day {
    background-position: 1128px -176px;
}
.heavy-snow-shower-night {
    background-position: 1182px -177px;
}
.light-rain-shower-night {
    background-position: 1126px -230px;
}
.light-rain-shower-day {
    background-position: 1072px -230px;
}

.drizzle {
    background-position: 1072px -230px;
}

```

```

}
.light-rain {
    background-position: 1019px -230px;
}
.sunset {
    background-position: 967px -230px;
}

/*Wind direction arrow icon classes*/
.S {
    background-position: 0 -18px;
}
.SSW {
    background-position: -53px -18px;
}
.SW {
    background-position: -105px -18px;
}
.WSW {
    background-position: -178px -18px;
}
.W {
    background-position: -213px -147px;
}
.WNW {
    background-position: 1px -78px;
}
.NW {
    background-position: -55px -76px;
}
.NNW {
    background-position: -109px -76px;
}
.N {
    background-position: -163px -75px;
}
.NNE {
    background-position: -219px -76px;
}
.NE {
    background-position: -3px -145px;
}
.ENE {
    background-position: -74px -146px;
}
.E {
    background-position: -154px -147px;
}
.ESE {
    background-position: -76px -220px;
}
.SE {
    background-position: -150px -222px;
}
.SSE {
    background-position: -296px -221px;
}
}

```

app/components/search-input/search-input.component.ts

```
/**
 * @author James Dinan
 * @date January 2017
 * SearchLocationsComponent - Renders location search results based on the user input.
 */
import {Component} from "@angular/core";
import {SelectedLocation} from "../../../services/selected-location.service";

@Component({
  moduleId: 'app/components/search-input/',
  selector: 'search-input',
  template: `
    <input [(ngModel)]="searchTerm" (ngModelChange)="setSearchLocationTerm()" class="search-input"
    autocomplete="off" type="text" name="search" placeholder="Search" autofocus />
    <style>
      input {
        background: none; border: none; color: white;
        outline: none;
        font-size: 18px;
        font-weight: bold;
        height: 30px;
        margin-left: 15px;
        text-decoration-color: white;
      }
      input::-webkit-input-placeholder {
        color: rgba(0, 183, 218, 0.36); font-size: 18px; font-weight: normal;
      }
    </style>
  `, providers: []
})
export class SearchInputComponent {
  public searchTerm = '';
  constructor(public selectedLocation: SelectedLocation) {}
  /**
   * Sets the search location term based on the user input.
   */
  setSearchLocationTerm(){
    this.selectedLocation.setSearchLocationTerm(this.searchTerm);
  }
}
```


app/components/previous-locations/previous-locations.component.ts

```
/**
 * @author James Dinan
 * @date January 2017
 * PreviousLocationsComponent
 * Renders the previously searched forecast locations and handles click event.
 */
import {Component, Input} from "@angular/core";
import {SelectedLocation} from "../../services/selected-location.service";

@Component({
  moduleId: 'app/components/previous-locations/',
  selector: 'previous-locations',
  styleUrls: ['previous-locations.css'],
  template: `
    <ul>
      <li (click)="setLocation({'id': '350029', 'name': 'A\\'Chralaig', 'lat': '57.1825', 'lon': '-5.1542',
        'region': 'he', 'regionID': '501'})">A'Chralaig</li>
      <li (click)="setLocation({'id': '310047', 'name': 'Cheltenham', 'lat': '51.8972', 'lon': '-2.0742',
        'region': 'sw', 'regionID': '513'})">Cheltenham</li>
      <li (click)="setLocation({'id': '310034', 'name': 'Bournemouth', 'lat': '50.7187', 'lon': '-1.8794',
        'region': 'sw', 'regionID': '513'})">Bournemouth</li>
      <li (click)="setLocation({'id': '371352', 'name': 'Black Fell', 'lat': '54.7932', 'lon': '-2.5472',
        'region': 'nw', 'regionID': '507'})">Black Fell</li>
    </ul>
  `,
  providers: []
})

export class PreviousLocationsComponent {
  constructor(private selectedLocation: SelectedLocation) {}

  /**
   * Calls service function to set the selectedLocation.
   * @param {object} location
   */
  setLocation(location) {
    this.selectedLocation.setLocation(location);
  }
}
```

app/components/previous-locations/previous-locations.css

```
/*
 * @author James Dinan
 * @date February 2017
 * Styling for the PreviousLocationsComponent.
 */
ul {
  flex: 1;
  list-style: none;
  padding: 0;
  margin: 0;
}
ul li {height: 40px; line-height: 25px; padding: 6px; border-bottom: solid 1px #f1f1f1;}
ul li:hover {background: #a9dbe1;}
```

app/components/search-locations/search-locations.component.ts

```
/**
 * @author James Dinan
 * @date January 2017
 * SearchLocationsComponent
 * Renders location search results based on the user input.
 */
import {Component} from "@angular/core";
import {SelectedLocation} from "../../../services/selected-location.service";
import {LocationService} from "../../../services/location.service";
import {ForecastLocation} from "../../../services/location";

@Component({
  moduleId: 'app/components/search-locations/',
  selector: 'search-locations',
  styleUrls: ['search-locations.css'],
  template: `
    <ul>
      <li class="search-location" *ngFor="let location of locations | filter:{name: (filterText.length > 2) ?
        filterText : '%'}" (click)="setLocation(location)">{{location.name}}</li>
    </ul>
  `,
  providers: [LocationService]
})
```

```

export class SearchLocationsComponent {
  constructor(private selectedLocation: SelectedLocation, private LocationsService:LocationService) {
    selectedLocation.searchLocationTerm$.subscribe(term => this.filterText = term);
  }
  public locations:ForecastLocation[] = [];
  public locationsError:String = '';
  public filterText:String = '';

  /**
   * Initialises SearchLocationsComponent - makes service calls to retrieve locations.
   */
  ngOnInit():any {
    this.getLocations();
  }

  /**
   * Calls service functions to retrieve locations data from Met Office DataPoint API.
   * Sets class variables and error state accordingly.
   */
  getLocations(){
    this.LocationsService.getLocations().subscribe(
      res => {
        this.locations = res;
      },
      error => this.locationsError = <any>error
    );
  }

  /**
   * Calls service function to set the selectedLocation.
   * @param {object} location
   */
  setLocation(location) {
    this.selectedLocation.setLocation(location);
  }
}

```

app/components/search-locations/search-locations.css

```

/*
 * @author James Dinan
 * @date February 2017
 * Styling for the SearchLocationsComponent.
 */
ul {
  flex: 1; list-style: none; padding: 0 0 10px 0; margin: 0;
}
ul li {
  height: 40px; line-height: 25px; padding: 6px 6px 6px 10px; border-bottom: solid 1px #f1f1f1;
  width: 100%; text-overflow: ellipsis; white-space: nowrap; overflow: hidden;
}
ul li:hover {background: #a9dbe1;}

```

app/pipes/search-filter.pipe.ts

```

/**
 * @author James Dinan
 * @date January 2017
 * FilterPipe
 * Filters array results for the given property and value.
 */
import {Pipe, PipeTransform} from '@angular/core';
@Pipe({
  name: 'filter'
})
export class FilterPipe implements PipeTransform {
  transform(items: any, filter: any): any {
    if (filter && Array.isArray(items)) {
      let filterKeys = Object.keys(filter);
      return items.filter(item =>
        filterKeys.reduce((memo, keyName) =>
          (memo && new RegExp(filter[keyName], 'gi').test(item[keyName])) || filter[keyName] === "", true));
    } else {
      return items;
    }
  }
}

```

9.10 Appendix 10: CSS Non-Optimised Branch

The following CSS changes were made to the app.shell.css file for the testing of proposed optimisation techniques against traditional animations

app/shell/app.shell.css

```
#search-panel {
  width: 100%;
  height: 100%;
  position: absolute;
  z-index: 30;
  top: 56px;
  left: 0;
  background: #FAFAFA;
  overflow: auto;
}

.search-panel-show {
  visibility: visible;
  opacity: 1;
  transition: opacity 0.3s, visibility 0.3s;
}
.search-panel-hidden {
  visibility: hidden;
  opacity: 0;
  transition: opacity 0.3s, visibility 0.3s;
}

.side-nav {
  background-color: white;
  display: flex;
  flex-direction: column;
  height: 100%;
  max-width: 400px;
  position: relative;
  left: -102%;
  width: 90%;
  padding: 20px;
}

.side-nav--animatable .side-nav {
  transition: 0.2s cubic-bezier(0,0,0.3,1);
  left: -102%;
}

.side-nav--visible.side-nav--animatable .side-nav {
  transition: 0.2s cubic-bezier(0,0,0.3,1);
  left: -102%;
}

.side-nav--visible .side-nav {
  transform: none;
  transition: 0.2s cubic-bezier(0,0,0.3,1);
  left: 0;
}
```

9.11 Appendix 11: Web Worker Branch

app/services/web-worker.service.ts

```
/**
 * @author James Dinan
 * @date March 2017
 * WebWorker Service.
 * Allows any non dom-manipulation function to be run inside a worker thread.
 */
import {Injectable} from '@angular/core';

@Injectable()
export class WebWorkerService {
  private promiseToWorkerMap = new WeakMap<Promise<any>, Worker>();
  private worker;

  /**
   * Initialises the worker, running the passed in function
   * @param {function} workerFunction
   */
  init<T>(workerFunction: (input: any) => T): void {
    let url = this.createWorkerUrl(workerFunction);
    this.worker = new Worker(url);
  }

  /**
   * Creates an blob with the passed in workerFunction.
   * Allows any function to be passed to the service and run in a web worker.
   * @param {function} workerFunction
   * @returns {string}
   */
  private createWorkerUrl(workerFunction: Function): string {
    let resolveString = workerFunction.toString(),
        webWorkerTemplate = `
      self.addEventListener('message', function(e) {
        postMessage((${resolveString})(e.data));
      });
    `;
    blob = new Blob([webWorkerTemplate], { type: 'text/javascript' });
    return URL.createObjectURL(blob);
  }
}
```

```

/**
 * Sends postMessage to the worker thread with the passed in data.
 * Returns Promise to calling component.
 * @param {object} data
 * @returns {Promise<T>}
 */
runWorker(data?: any): Promise<any> {
    let promise = this.createPromiseForWorker(this.worker, data),
        removePromise = this.removePromise(promise);
    this.promiseToWorkerMap.set(promise, this.worker);
    promise.then(removePromise).catch(removePromise);
    return promise;
}

/**
 * Creates a promise with message and error listeners and calls postMessage.
 * @param {object} worker
 * @param {object} data
 * @returns {Promise<T>}
 */
private createPromiseForWorker<T>(worker: Worker, data: any) {
    return new Promise<T>((resolve, reject) => {
        worker.addEventListener('message', (event) => resolve(event.data));
        worker.addEventListener('error', reject);
        worker.postMessage(data);
    });
}

/**
 * Removes promise after being returned.
 * @param {Promise} promise
 * @returns {function(any): any}
 */
private removePromise<T>(promise: Promise<T>) : (input: any) => T {
    return (event) => {
        this.promiseToWorkerMap.delete(promise);
        return event;
    };
}
}

```

app/pipes/search-filter.pipe.ts

```
/**
 * @author James Dinan
 * @date January 2017
 * FilterPipe
 * Filters array results for the given property and value.
 */
import {Pipe, PipeTransform} from '@angular/core';
import {WebWorkerService} from '../services/web-worker.service';

@Pipe({
  name: 'filter'
})
export class FilterPipe implements PipeTransform {
  constructor(private _webWorkerService: WebWorkerService){
    this._webWorkerService.init(this.searchFilter);
  }
  private data;

  transform(items: any, filter: any): any {
    var input = {items: items, filter: filter};
    var promise = this._webWorkerService.runWorker(input);
    promise.then(result => this.data = result);
    return this.data;
  }

  searchFilter (input) {
    let filter = input.filter, items = input.items;
    if (filter && Array.isArray(items)) {
      let filterKeys = Object.keys(filter);
      return items.filter(item =>
        filterKeys.reduce((memo, keyName) =>
          (memo && new RegExp(filter[keyName], 'gi').test(item[keyName])) || filter[keyName] === "", true));
    } else {
      return items;
    }
  }
}
```


9.12 Appendix 12: GraphQL Branch

The following appendix contains the main code created as part of the graphql branch. The files modified or added from the dev branch are included. The following dependencies were installed: "request": "2.81.0", "graphql": "0.9.2", "express-graphql": "0.6.4", "node-fetch": "1.6.3".

server.js

```
/**
 * Node Server startup script.
 * Sets up the express routing for the original proxy api and an additional graphql route.
 * This graphql route uses the schema specified in datapoint.schema.js to return the requested data.
 */
var express = require('express'), app = express(),
    graphqlHTTP = require('express-graphql'), locationSchema = require('./datapoint.schema'),
    request = require('request'), port = (false) ? process.env.PORT : 8080, directory = (false) ? '/release' : '';

/**
 * Sets the static directory from which to host the files based on presence of Amazon port environment variable.
 * Proxy to ensure requests to the Met Office DataPoint API are returned through server and HTTPS connection.
 */
app.use(express.static(__dirname + directory, { redirect : false }));
app.use('/proxy', function(req, res) {
    var url = req.url.replace('/?url=', '');
    req.pipe(request(url)).pipe(res);
});

/**
 * Sets the datapoint key as a global param for use by the schema file,
 * Specifies the graphql endpoint.
 */
GLOBAL.dataPointKey = '0418f3e9-cdb2-4ad8-8b62-ea2f1a422755';
app.use('/graphql', graphqlHTTP({
    schema: GLOBAL.schema,
    graphiql: true
}));
app.listen(port, function(){
    console.log('Server running on port: ' + port);
});
exports = module.exports = app;
```

datapoint.schema.js

```
/**
 * @author James Dinan
 * @date March 2017
 * GraphQL Schema
 * Sets up the query fields and makes requests to the Met Office DataPoint API
 * Adapted from prototype version by Andrew Poyntz (https://github.com/AndrewPoyntz/datapoint-graphql-server)
 * to include data calls to daily forecast service and combine results.
 */
var graphql = require('graphql'),
    fetch = require('node-fetch'),
    moment = require('moment'),
    GraphQLObjectType = graphql.GraphQLObjectType,
    GraphQLString = graphql.GraphQLString,
    GraphQLSchema = graphql.GraphQLSchema,
    GraphQLList = graphql.GraphQLList;

/**Fetch requests - following functions are equivalent to the rest API calls within the angular service files.
 * location.service.ts and forecast.service.ts.*/
/*-----*/
/**
 * Get request to retrieve the 3-Hourly forecast data from the Met Office API for a specific siteID.
 * @param {string} id - locationID
 * @returns {object}
 */
var get3HourlyForecast = function (id) {
  return fetch('http://datapoint.metoffice.gov.uk/public/data/val/wxfcs/all/json/' + id + '?res=3hourly&key=' +
    GLOBAL.dataPointKey)
    .then(res => res.json())
    .then(json => {
      return json.SiteRep.DV
    })
    .then((data) => {
      return {
        issueTime: data.dataDate,
        days: data.Location.Period
      }
    });
};
```

```

/**
 * Get request to retrieve the daily forecast data from the Met Office API for a specific siteID.
 * @param {string} id - locationID
 * @returns {object}
 */
var getDailyForecastData = function (id) {
  return fetch('http://datapoint.metoffice.gov.uk/public/data/val/wxfcs/all/json/' + id + '?res=daily&key=' +
GLOBAL.dataPointKey)
    .then(res => res.json())
    .then(json => {
      return json.SiteRep.DV
    })
    .then((data) => {
      return {
        issueTime: data.dataDate,
        days: data.Location.Period
      }
    })
};

/**GraphQL object definition schemas. Defines the fields possible within graphql queries to the service.*/
/-----*/
/**
 * Main DataPoint definition, with locations list.
 * @type {graphql.GraphQLObjectType}
 */
const Datapoint = new GraphQLObjectType({
  name: 'Datapoint',
  description: '3-Hourly and Daily Forecast data for UK Locations',
  fields: () => ({
    location: {
      description: 'Forecast Locations',
      type: new GraphQLList(LocationType),
      args: {
        id: {
          type: GraphQLString
        }
      },
      resolve: (_, args) => {
        return [{id: args.id}];
      }
    }
  })
});

```

```

/**
 * ForecastDays - splits the 3Hourly forecast timesteps by days.
 * @type {graphql.GraphQLObjectType}
 */
const ForecastDays = new GraphQLObjectType({
  name: 'ForecastDays',
  fields: ()=>({
    issueTime: {
      type: GraphQLString
    },
    days: {
      type: new GraphQLList(ForecastDates),
      resolve: data => sortForecastDates(data.days)
    }
  })
});

/**
 * ForecastDaily - splits the overall daily forecast by days.
 * @type {graphql.GraphQLObjectType}
 */
const ForecastDaily = new GraphQLObjectType({
  name: 'ForecastDaily',
  fields: ()=>({
    issueTime: {
      type: GraphQLString
    },
    days: {
      type: new GraphQLList(ForecastDailyDays),
      resolve: data => sortForecastDailyDays(data.days)
    }
  })
});

```

```

/**
 * Splits the forecast days into the individual timestep dates.
 * @type {graphql.GraphQLObjectType}
 */
const ForecastDates = new GraphQLObjectType({
  name: 'ForecastDates',
  fields: ()=>({
    date: {type: GraphQLString},
    timesteps: {
      type: new GraphQLList(ForecastTimesteps),
      resolve: data => {
        return sortForecastTimesteps(data.date, data.dayData.Rep)
      }
    }
  })
});

/**
 * Defines the data that can be returned for the Daily Forecast object.
 * @type {graphql.GraphQLObjectType}
 */
const ForecastDailyDays = new GraphQLObjectType({
  name: 'ForecastDailyDays',
  fields: ()=>({
    date: {type: GraphQLString},
    uvIndex : {type: GraphQLString},
    weatherType : {type: GraphQLString},
    maxTemp : {type: GraphQLString}
  })
});

/**
 * Defines the data available to be returned by each of the forecast timesteps.
 * @type {graphql.GraphQLObjectType}
 */
const ForecastTimesteps = new GraphQLObjectType({
  name: 'ForecastTimesteps',
  fields: ()=>({
    time: {type: GraphQLString},
    uvIndex : {type: GraphQLString},
    weatherType : {type: GraphQLString},
    visibility : {type: GraphQLString},
    temp : {type: GraphQLString},
  })
});

```

```

        windSpeed : {type:GraphQLString},
        precipProbab : {type:GraphQLString},
        relativeHumidity : {type:GraphQLString},
        windGust : {type:GraphQLString},
        feelsLikeTemp : {type:GraphQLString},
        windDirection : {type:GraphQLString}
    })
  });

/**
 * Location data fields. Each location has its own specific attributes, and nested objects for
 * three-hourly and daily forecasts.
 * @type {graphql.GraphQLObjectType}
 */
const LocationType = new GraphQLObjectType({
  name: 'Location',
  description: 'A Location object defines a single site',
  fields: () => ({
    id: {
      type: GraphQLString
    },
    ThreeHourlyForecasts: {
      type: ForecastDays,
      resolve: location => {
        return get3HourlyForecast(location.id)
      }
    },
    DailyForecasts: {
      type: ForecastDaily,
      resolve: location => {
        return getDailyForecastData(location.id)
      }
    }
  })
});

/**Helper functions for sorting data.*/
/*-----*/
/**
 * Sort the 3-Hourly forecast timesteps.
 * @param {string} dayDate
 * @param {object} data
 * @returns {Array}

```

```

*/
var sortForecastTimesteps = function (dayDate, data) {
  var timesteps = [], i, timestep;
  for (i = 0; i < data.length; i++) {
    timestep = data[i];
    timesteps.push({
      time: moment(dayDate).add(timestep['$'], 'minutes').toISOString(),
      uvIndex:timestep.U,
      weatherType:timestep.W,
      visibility:timestep.V,
      temp:timestep.T,
      windSpeed:timestep.S,
      precipProbab:timestep.Pp,
      relativeHumidity:timestep.H,
      windGust:timestep.G,
      feelsLikeTemp:timestep.F,
      windDirection:timestep.D
    })
  }
  return timesteps;
};

/**
 * Sorts the full daily forecast by days
 * @param {array} data
 * @returns {Array}
 */
var sortForecastDailyDays = function (data) {
  var days = [], i, day, date;
  for (i = 0; i < data.length; i++) {
    day = data[i];
    date = moment(day.value.substring(0, day.value.length - 1) + 'T00:00:00.000Z').toISOString();
    day = day.Rep[0];
    days.push({
      date: date,
      uvIndex:day.U,
      weatherType:day.W,
      maxTemp:day.Dm
    })
  }
  return days;
};

```

```

/**
 * Sorts the 3-Hourly forecasts by days.
 * @param data
 * @returns {Array}
 */
var sortForecastDates = function (data) {
  var days = [], i, day, date;
  for (i = 0; i < data.length; i++) {
    day = data[i];
    date = moment(day.value.substring(0, day.value.length - 1) + 'T00:00:00.000Z').toISOString();
    days.push({
      date: date,
      dayData: day
    })
  }
  return days;
};

//Export schema.
GLOBAL.schema = new GraphQLSchema({
  query: Datapoint
});

```

app/services/forecast.service.ts

In the forecast service, added the graphql query string, a single http.get call to the new api passing the query, and modified the extractData function all to the new data format.

```

private graphqlQueryString = `
{
  location(id : "310047") {
    id,
    DailyForecasts{
      issueTime,
      days{
        date, weatherType
      }
    }
    ThreeHourlyForecasts {
      days {date, timesteps {time, weatherType, temp, windSpeed, windGust, windDirection}}
    }
  }
}
`;

```



```

/**
 * Replacing dependant API requests with a single graphql query.
 * @returns {Observable<R>}
 */
getGraphQLQuery() {
    return this.http.get('/graphql?query=' + this.graphQLQueryString)
        .map((res) => this.extractData(res))
        .catch((error) => this.handleError());
}

/**
 * Extracts the 3-hourly forecast data.
 * @param {Response} res
 * @returns {Array}
 */
private extractData(res:Response) {
    let forecastData = res.json(),
        forecast = [],
        updated = new Date(res.json().data.location[0].DailyForecasts.issueTime);
    if (forecastData.data.location[0].ThreeHourlyForecasts.days instanceof Array) {
        for (let index in forecastData.data.location[0].ThreeHourlyForecasts.days) {
            let day = forecastData.data.location[0].ThreeHourlyForecasts.days[index],
                timesteps = this.extractTimesteps(day),
                dayWeatherType = res.json().data.location[0].DailyForecasts.days[index].weatherType;
            forecast.push({
                updated: updated.getHours() + ":" + (updated.getMinutes() < 10 ? '0' : '') + updated.getMinutes(),
                date: day.date,
                day: this.getFormattedDay(new Date(day.date).getDay()),
                weatherType: dayWeatherType,
                weatherTypeIconClass: this.getWeatherTypeIconClass(dayWeatherType),
                timesteps: timesteps
            });
        }
    }
    return forecast;
}

/**
 * Extracts and processes the timestep data from a given day object.
 * @param {object} day
 * @returns {{temperature: number}[]}
 */

```

```

private extractTimesteps(day:any):any {
  let daySteps = day.timesteps,
      timesteps = [];
  if (daySteps instanceof Array) {
    for (let index in daySteps) {
      let timestep = daySteps[index];
      timesteps.push(
        {
          weatherType: timestep.weatherType,
          weatherTypeIconClass: this.getWeatherTypeIconClass(timestep.weatherType),
          windDirection: timestep.windDirection,
          windSpeed: timestep.windSpeed,
          windGust: timestep.windGust,
          temperature: timestep.temp,
          temperatureClass: this.getTemperatureColourClass(timestep.temp),
          time: '09'
        }
      );
    }
  }
  return timesteps;
}

```

app/components/home.component.ts

Replaced the nested API calls in home component to a single call to the GraphQL query.

```

/**
 * Service functions replaced with a single graphql call.
 * Sets class variables and error state accordingly.
 */
getTabularForecastData(){
  this.ForecastService.getGraphQLQuery().subscribe(
    res => {
      this.forecast = res;
      this.selectedForecast = this.forecast[0];
    },
    error => this.forecastError = <any>error
  );
}

```


Building scalable ‘Progressive Web Apps’ with comparable performance to that of native mobile applications.

James Dinan

Introduction

Advances in browser features, new APIs and changes in design principles have brought about the concept of Progressive Web Apps (PWAs). A PWA is built using current web development languages and hosted on a remote server, but aims to provide a web experience that emulates the look and appearance of native apps; launching from an icon on the home screen and working offline (Malovolta, 2016, p.2).

For PWAs to be a viable commercial alternative they must overcome the inherent performance issues associated with the web and be scalable; fitting with the requirements for maintainable code and having all the features deemed necessary by end users.

Importance of Performance

49% of individuals in a research survey (Hewlett Packard Enterprise, 2016, p.4) indicated that they expect a native app to perform actions within 2 seconds, and 48% have uninstalled a native app that regularly ran slowly. A PWA opening in full screen from an icon makes it impossible to differentiate from native apps and it would be expected to perform equally.

Benefits of PWAs over Native

- More portable.
- Cheaper to develop.
- Easily shareable via a URL.
- Quicker distribution of the latest fixes and features.

81% of users rated load speed as either the same or faster*

97% of users rated the speed a forecast returned as either the same or faster*.

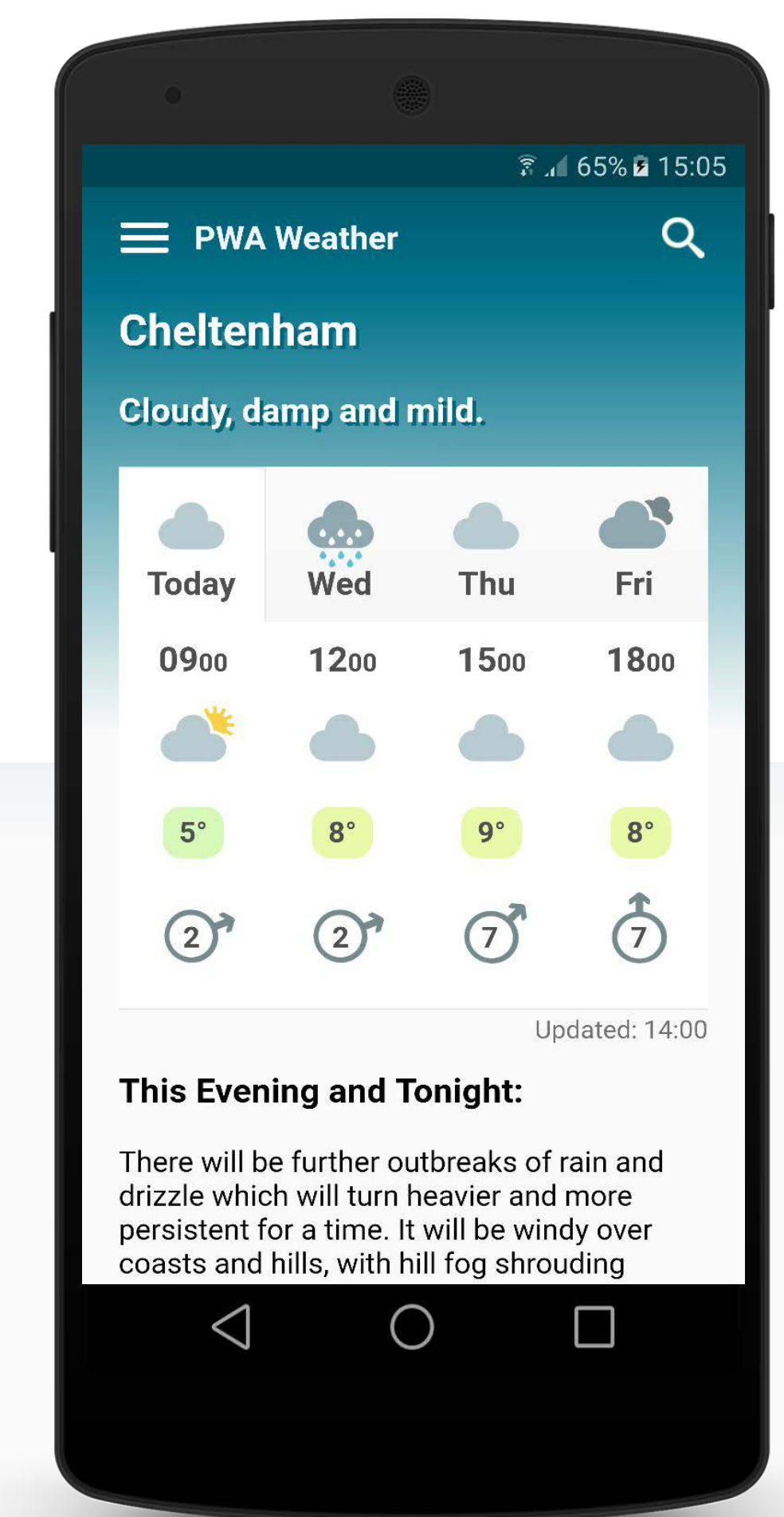
88% of users rated the performance of animations as either the same or smoother*.

*compared to competitor native applications

Development & Testing of PWA Weather

An objective of the paper was to create an app which could serve as a minimal viable product within industry, replicating the core functionality of competitor native apps. This was built using the Angular JavaScript framework and accessed data available from the Met Office DataPoint API.

The following optimisations were tested on a low and high-end device: Non-blocking JavaScript, Minification, Tree-shaking, CSS transform & will-change attributes, CSS pointer-events & opacity attributes, Web Workers and GraphQL. The test results, feedback from users and opinions of software professionals have been used to formulate best practice recommendations.



Best practice recommendations

- 1 A JavaScript framework facilitates an MVC architectural direction; ensures maintainability and easier unit testing.
- 2 Minify and tree-shake code. Minification quickened display of content by 57% on 3G. Tree-shaking reduced code by 12.4%.
- 3 Use an app shell with inline CSS and non-blocking scripts for a fast first paint. 43.6% improvement with non-blocking JS.
- 4 Apply CSS transforms and will-change for performant animations. FPS increased by 20 on the low-end device tested.
- 5 Use GraphQL instead of multiple REST APIs to prevent over-fetching. GraphQL reduced the time taken for data retrieval by 69%.

View from Industry

“ it costs huge amounts of money to write native apps across different devices...we have struggled with people picking up updates...if we want to change stuff we’ve got to keep backwards compatible versions of all our data feeds...So there’s massive advantages in my mind for a progressive one. ”
(Solutions Architect, Met Office, 2017).

Limitations and future work

Image processing is a limitation of JavaScript. The Met Office native app processes weather radar images for display on a map. A solution that shows such intensive processing is efficiently possible in a browser would be a positive future development to allow PWA’s to replicate native.

References

1. Malavolta, I. (2016). 'Beyond native apps: web technologies to the rescue! (keynote)', In *Proceedings of the 1st International Workshop on Mobile Development*, ACM, New York, NY, USA, 1-2
2. Hewlett Packard Enterprise. (2015). *Failing to Meet Mobile App User Expectations: A Mobile User Survey - Dimensional Research*. Available at: <http://go.saas.hp.com/techbeacon/apppulse-mobile-survey> (Accessed: October 2016)