JAY'S BLOG (HTTP://BLOG.JAYMEHTA.CO.UK/)
NHS DOCTOR, CLINICAL INFORMATICIAN, TECHNOPHILE

Posts

EPR Guide – An Open Source Web App

19th April 2019 Jay Mehta (http://blog.jaymehta.co.uk/author/jaymehta/) Leave a comment (http://blog.jaymehta.co.uk/2019/04/epr-guide-an-open-source-web-app/#respond)

I was recently given permission to make one of my projects at work (an EPR Guide Web App) open source, and in this post I hope to explain the project and the decisions I made so that other people can benefit from it.

https://github.com/jaymehta50/rfg-epr-guide (https://github.com/jaymehta50/rfg-epr-guide)

Disclaimer: This product is provided as-is, with no guarantees of compatibility or support. The authors cannot take any responsibility for any adverse consequences resulting from use of this product. Please check thoroughly before using in a production setting!

Background

I work at a London NHS Foundation Trust as Chief Medical Information Officer, and we recently went live with a brand new electronic patient record (EPR). As part of the go-live process, I wanted to create a guide for our staff, to disseminate information and act as a source of knowledge for them.

Given the nature of the job, I wanted to avoid paper! And with my past experience in web development and app design, I decided to turn my hand to a mobile app solution.

(http://blog.jaymehta.co.uk/wp-content/uploads/2019/04/www.jaymehta.co_.uk_rfg-

epr-guide_iPhone-6_7_8-Plus.png)

What is it?

This is a progressive web app (PWA) designed to inform staff about the EPR Go Live, with content on:

- Contextual information on EPR, the implementation project and the reasons for doing it.
- How staff can individually prepare for the EPR Go Live.
- Resources on how to use the EPR.
- Guidance on how the EPR will (or will not) change clinical workflows.
- Information on the Go Live weekend itself (known as Cutover), including a systems availability table and dynamic ward status table.
- Contact details.

Dependencies

For this project, I used:

- LAMP stack
- Domain name of https://royalfree.info/ (now expired)
- SSL Certificate
- CodeIgniter PHP Framework (https://codeigniter.com)
  - Simple, lightweight, easy to use PHP MVC framework
- MaterializeCSS (https://materializecss.com)
  - Responsive front-end framework, based on Google's Material Design (https://material.io).
  - If it's good enough for Google, it's good enough for me!

Whilst not a technical dependency – the Google Developer PWA Checklist (https://developers.google.com/web/progressive-web-apps/checklist) and Lighthouse (https://developers.google.com/web/tools/lighthouse/) audits were invaluable for me learning to create a PWA for the first time.

Why a PWA?

Primarily because I didn't have much time to get the project done – committing to native apps would have required separate iOS, Android and desktop development.

Also PWAs are an exciting new method of delivering app-like functionality over HTML, JS and CSS – with just one codebase that works regardless of device and screen size, I could focus on creating a fantastic user experience that was consistent for all staff. Check out the Google Developers PWA page (https://developers.google.com/web/progressive-web-apps/) for further reading.

The Build

PHP AND MYSQL

I used PHP code in a MVC framework to let me create lots of separate PHP view files, each containing a component written in HTML, formatted using MaterializeCSS styling. I could then use a controller PHP file to stitch all of these component views into one single page of HTML for the user.

This dynamic approach was fantastic, as when combined with data held in the MySQL database, I could quickly generate lots of instances of one component, each with different contents determined by the database.

For example – the How To guides and the Workflows are held as data in a MySQL database, and the Controller and Model simply use one view file over and over again with each row of data from the database, to generate multiple HTML components that get stitched into the overall package sent to the user.

In addition – I was able to grant colleagues access to the database so that they could create and insert data directly, confident in the knowledge that the front-end would dynamically turn their plain data into presentable content.

JAVASCRIPT

After playing around with a few Javascript frameworks I didn't find anything particularly easy and lightweight, most of them had way too many features relative to what I needed and I was conscious of minimising the app's size. As a result I decided to write the javascript entirely myself.

Internal Navigation:

- Unlike a traditional website where each page is its own HTML document, in a PWA all the "pages" are containers in the same HTML document and the inactive ones are just hidden (a single-page application, or SPA).
- So I changed all the internal navigation links to use a custom data-route attribute instead of the traditional href attribute, then created a router in javascript to interpret these clicks and show the corresponding container (whilst hiding all the rest).

Offline availability:

- I created a Service Worker javascript file, which more and more browsers now recognise and use to cache files offline so that the website still works even without an internet connection.
- This in my opinion is the main reason that the website feels like an "app".
- Checkout the Google Developers Page on Service Workers (https://developers.google.com/web/fundamentals/primers/service-workers/) for further info.

Install prompts:

- One advantage of a PWA is that it can be "installed" to a user's mobile Home Screen.
- This is not really an install, it's more like adding a bookmark to your home screen, but it gives users a sense that this is a native app and they therefore treat it as if it is available offline (which is what I want).
- Chrome on Android has functions specifically for this purpose, whereas iOS does not, so I created a banner specifically for iOS and tapped into the Chrome for Android methods as described here (https://developers.google.com/web/fundamentals/app-install-banners/#defer_or_cancel).

Data storage:

- One feature of the app is that staff can "get ready" for EPR by marking certain tasks as complete.

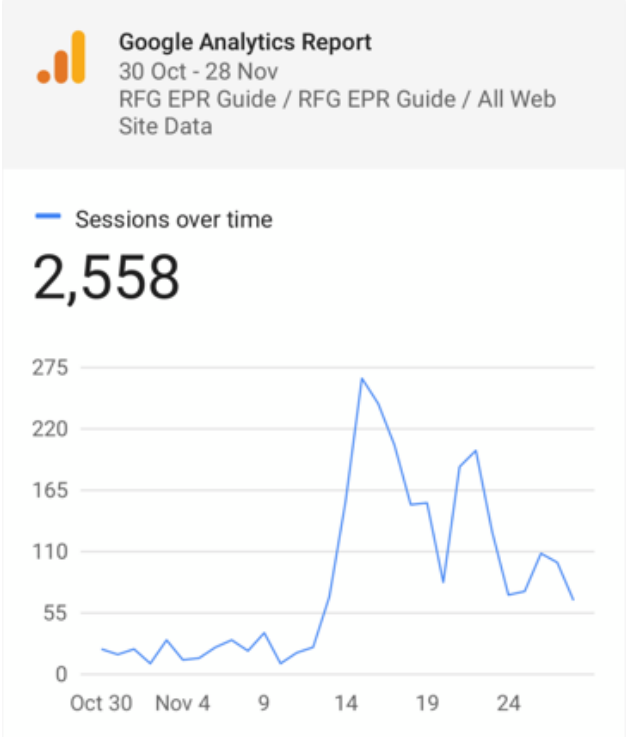
- As a result there must be some way to retain that data so that when the user opens the app again, their “completed” tasks do not suddenly reset.
- I was initially tempted to ask users to login, and then store this data in the server’s MySQL database.
- However I decided against this for two reasons:
  - I don’t have any other use for this data – and from a privacy point of view, we shouldn’t collect any data unless we absolutely have to. I don’t believe in invading users’ privacy unless necessary for the functionality they want.
  - Asking users to login will disrupt their flow, and cause an unnecessary drop in engagement.
- So instead, I wrote a javascript function to save this data locally, and accept that staff using multiple devices or staff who disable local storage on their devices see a suboptimal experience.

Finally, I also used Javascript to implement some user-interface feature such as searches and to progress through workflow questions – see source code comments for further info.

Did it work?

I think so!

Anecdotally I heard lots of good things from users, which is always nice! But I also measured analytics which showed the usage below. Given we had an audience of approx 3000 staff, I’m very happy that the web app reached the usage below:



As always – please feel free to reach out with any comments or suggestions, and check out the source code on Github at:

https://github.com/jaymehta50/rfg-epr-guide (https://github.com/jaymehta50/rfg-epr-guide)

(/#facebook) (/#twitter) (/#email) (/#linkedin) (/#whatsapp) (/#copy_link)

Posted in: Clinical UX Edit (http://blog.jaymehta.co.uk/wp-admin/post.php?post=165&action=edit) (http://blog.jaymehta.co.uk/category/clinical-ux/), Open source software (http://blog.jaymehta.co.uk/category/clinical-ux/open-source-software/)
Filed under: EPR Guide (http://blog.jaymehta.co.uk/tag/epr-guide/), Open source (http://blog.jaymehta.co.uk/tag/open-source/), Progressive Web App (http://blog.jaymehta.co.uk/tag/progressive-web-app/), PWA (http://blog.jaymehta.co.uk/tag/pwa/), Single Page Application (http://blog.jaymehta.co.uk/tag/single-page-application/), Web App

(http://blog.jaymehta.co.uk/tag/web-app/)

← My EHR Design
(http://blog.jaymehta.co.uk/2019/04/my-ehr-design/)

LEAVE A REPLY

Logged in as Jay Mehta (http://blog.jaymehta.co.uk/wp-admin/profile.php). Log out? (http://blog.jaymehta.co.uk/wp-login.php?action=logout&redirect_to=http%3A%2F%2Fblog.jaymehta.co.uk%2F2019%2F04%2Fepr-guide-an-open-source-web-app%2F&_wpnonce=a9b7360bbb)

Comment

POST COMMENT

This site uses Akismet to reduce spam. Learn how your comment data is processed (https://akismet.com/privacy/).

ARCHIVES

April 2019 (http://blog.jaymehta.co.uk/2019/04/)

March 2019 (http://blog.jaymehta.co.uk/2019/03/)

CATEGORIES

About Me (http://blog.jaymehta.co.uk/category/about-me/)

Clinical UX (http://blog.jaymehta.co.uk/category/clinical-ux/)

Open source software (http://blog.jaymehta.co.uk/category/clinical-ux/open-source-software/)

IMAGE ISSUES

The images listed below are being resized to fit a container. To avoid serving oversized or blurry images, try to match the images to their container sizes.

Oversized
1 504 x 1024px 252 × 512px
This image is too large for its container. Adjust the image dimensions to 252 x 512px for optimal results.
2 800 x 1024px 366 × 485px
This image is too large for its container. Adjust the image dimensions to 366 x 485px for optimal results.
3 800 x 1024px 366 × 485px

This image is too large for its container. Adjust the image dimensions to 366 x 485px for optimal results.

Note: It's not always easy to make this happen, fix up what you can.