WEB AND MULTI-PLATFORM MOBILE APP AT ELETTRA

L. Zambon*, A. I. Bogani, S. Cleva, E. Coghetto, F. Lauro, Elettra-Sincrotrone Trieste S.C.p.A., Trieste, Italy
M. De Bernardi, University of Trieste, Trieste, Italy

Abstract

A few apps have been recently developed at Elettra Sincrotrone Trieste. The main requirements are the compatibility with the main mobile device platforms and with the web, as well as the "mobile-first" user interface approach. We abandoned the possibility of developing native apps for the main mobile OSs. There are plenty of libraries and frameworks for the development of modern cross platform web/mobile applications. In this scenario the choice of a particular set of libraries is crucial. In this paper we will discuss the motivation of our choice trying to compare it with the other possibilities in regard to our particular use cases, as well as the first applications developed.

INTRODUCTION

In late 2016 we developed a hybrid app called elettrApp, this app was based mainly on Apache Cordova and jQuery. In May 2017 started a 12 months project called PWMA (Platform for Web and Mobile Applications) based mainly on WebSockets and React Native. Although we are still at the beginning, the results already obtained are very encouraging and we are confident to reach much better results in the near future.

ELETTRAPP

Requirements

- **Multiplatform** We considered the possibility to build a hybrid app more attractive than a native app for Android because hybrid apps can run on all the main mobile platforms and on the web.
- **Fast development** Most of the development was done by a bachelor's degree student as his thesis. The time available wasn't much longer than two months. We expected to develop an app equivalent to a single synoptic panel already implemented as a native GUI (Graphic User Interface) written in C++ and Qt, but hybrid apps development was so quick that allowed to include a few other screens, one of them much more complicated.
- **All in one application** Our users asked explicitly for a unique mobile app as an interface for all tasks.

Technology

- **Apache Cordova** allows to develop hybrid apps that is: "Hybrid apps embed a mobile web site inside a native app. [...] This allows development using web technologies [...] while also retaining certain advantages of native apps (e.g. direct access to device hardware, offline operation, app store visibility)."[1]. We used an excellent documentation [2] which allowed us to be productive almost immediately. Apache Cordova runs on the mobile browser, but the browser is hidden to the user so that the application looks like a native application. The browser implies some inefficiency in comparison to native apps, but the only sector interdicted is interactive gaming. An other important feature of hybrid apps is the possibility to download from the web not only data but also templates (in form of HTML (HyperText Markup Language) and JavaScript files). This makes an app much more expandable and flexible [3].
- **jQuery** is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML [4]. jQuery contributed significantly to speed up the development time in respect to vanilla JavaScript [5]. jQuery is responsible of connecting asynchronously to a REST (REpresentational State Transfer) server. We used only client to server connections (polling).
- **Bootstrap** is a free and open-source front-end web framework for designing websites and web applications. It contains HTML- and CSS-based (Cascading Style Sheet) design templates [6]. Bootstrap implements a “mobile first” design [7]. From the developer point of view, using Bootstrap requires little more effort than writing basic HTML, but the user experience is greatly improved.

- **Ionic** We tried to use Ionic [8] but the benefits from this framework didn't come quickly enough, so we aborted this part of our development. This was due to the fact that Ionic require AngularJS [9] which is a JavaScript framework completely different from jQuery.

Architecture

The first screen is a basic starter composed by a button for each task plus a tick which makes the task chosen the default.

The tasks are: synoptic status (for 2 accelerators and a complex system), cAstor, a starter administration tool similar to TANGO Astor (for 3 accelerator domains) and a shift calendar (for 2 groups).

- **Synoptic status** A common pattern is shared by all synoptic status. Each synoptic screen is composed by one or two charts. The charts are always on top because our users asked to put them in evidence. Charts are embedded in an <iframe> tag using

* email address lucio.zambon@elettra.eu

eGiga2m, a tool already used at Elettra [10]. This is a trivial way to use other applications as a component in a new one. This technique reduced dramatically the development time. The price to pay is a certain inefficiency at run time, but in our case it was acceptable. Below the charts there are a few boxes which are by default aligned vertically and closed on small devices; on large devices the same boxes are opened by default and are inserted in 2 or 4 columns. Each box contains a few data, each with its label. These data was retrieved periodically from the REST server. During the development, a few tests have been carried out to find the best way to group the data together and find the optimal period of the timer used to poll them.

- **cAstor** is a modified version of TANGO Astor [11]. The tree structure of Astor was implemented but most menus were neglected or simplified. When a device server is switched on or off, a modal containing an <iframe> is opened. In this <iframe> there is an authentication form provided by an external service secured with TLS (Transport Layer Security). We created a new view which allows to monitor only the stopped device servers. Another extension consisted in browsing up to the attribute level and installing an alarm when crossing a configurable threshold. But this feature was aborted because it was based on a very consuming polling system.
- **Shift calendars** are simple adaptive tables configured for our operators and for our machine physicists.

PWMA

PWMA started as a mini project of Elettra Sincrotrone Trieste aimed to obtain a unified web and app platform for our institute. Anyway we are open to possible extensions and collaborations. All source code is hosted at https://gitlab.com/PWMA/

Requirements

- **Multiplatform** PWMA must support the main evergreen browsers and the main mobile app systems.
  It is acceptable that some old version of browsers and of mobile OS or negligible market-share browsers are not supported. It is almost not acceptable to depend on any configuration of browsers. Mobile app should perform as much as possible like native app.
- **Event driven** The client-server communication is not driven by a periodic refresh triggered by the client, but it is asynchronous on both sides. This should minimize the traffic on the web and minimize the latency. SSE (Server-sent events) [12] with HTTP/2 (Hypertext Transfer Protocol) offers auto-reconnect, multiplexing on the same TCP/IP (Transmission Control Protocol / Internet Protocol) connection and unidirectionality; this feature together with a REST interface allows a strategy of type CQRS (Command Query Separation) [13]; on the other hand WebSockets [14] are supported by more libraries, among which React Native.
  The high efficiency provided by event driven data transfer is a stimulus for performing technology on client side.
- **Components** should facilitate the automation of composition of a large number of pages, the composition of new pages should be eased by a designer. WebComponents are a standard provided by W3C (World Wide Web Consortium) and as such it will be natively supported by all evergreen browsers. There are other implementations of components which should be considered as a trade-off with high efficiency.
- **All in one application** the number of screens till now is about one hundred, but in future it may be much larger. Our goal is to have the possibility to scale up to several thousands screens. We suppose that the only data structure fit to manage these numbers is a tree-like structure. A tree is very uncommon on mobile apps, but we opted for it also because our users are accustomed to it.

Technology

Some libraries/frameworks had been evaluated but proved not satisfactory.

- **AngularJS** doesn't support components [15]
- **Polymer** isn’t fully supported by browsers and it has low efficiency with polyfill.

React and React Native looked to us as the best compromise available.

- **React** [16] is a JavaScript library created at Facebook in 2011 and open-sourced in 2013. React implements a state machine with well defined state transitions. React is declarative and comes with a custom language JSX (JavaScript Syntax eXtension). React doesn't support WebComponents, but proprietary components and Virtual DOM (Document Object Model) instead of Shadow DOM.
- **React Native** [17] shares the same architecture of React but creates native iOS, Android and UWP (Universal Windows Platform) applications. In React Native JavaScript code runs in a separated thread and generates elements by calling asynchronously platform native procedures. So most of the computation time is managed by native high efficiency procedures. React Native supports WebSockets and doesn't support SSE (Server Sent Events) over HTTP/2. This is not ideal for us, we would prefer standard components and standard language but React and React Native have encountered a huge success which translates into large support (for instance several books [18, 19] even for advanced developers [20], many pages on github.com and stackoverflow.com). React Native

shares with React almost everything except the DOM. React Native is growing quickly, every month there is a new release with exciting new features, but occasionally with a few compatibility problems which take time to be solved.

We used a lint checker with AirBnB configuration, but we did some exceptions to this set of rules [21].

Architecture

There are two main parts: a pool of servers which are mainly event driven except for a few secondary tasks are traditional servers (green in Fig. 1) and clients which interact mainly asynchronously with servers (light blue in Fig. 1).

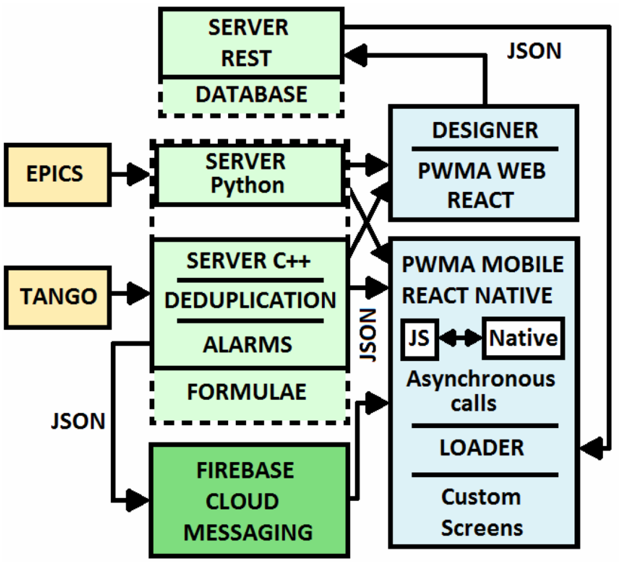


Figure 1: Layout of PWMA. All servers developed by us are in light green, external servers are in dark green, clients are in light blue, parts included in dashed lines are not jet implemented.

Servers

Two main servers connect PWMA with Tango and EPICS on one side and to WebSocket through a JSON (JavaScript Object Notation) [22] API (Application Programming Interface) on the other side. This API was designed to be as generic as to give the possibility to switch Control System and communication protocol from WebSocket to SSE.

The first server is written in C++ and connects asynchronously to Tango, it performs decoupling of the requests, i.e. if two clients subscribe to the same variable the control system will receive only one subscription. Attributes not configured to support events are polled very slowly. There is a limited implementation of alarms, only very simple formulae are evaluated and each generated alarm is passed to FCM (Firebase Cloud Messaging) [23].

The second server borrows a Python library [24]. It connects asynchronously to EPICS by using monitors and to the rest of PWMA by adapting a WebSocket library [25]. It is still a proof of concept, but it is complete, except the error management.

The variables coming from both control systems can be mixed because we use FQDN (Fully Qualified Domain Name) and we plan to merge the two main servers in a unique server supporting both EPICS and Tango and any other control system which will be supported in the future.

Clients

The client part is structured in three levels: Components, Screens and Starter.

Components

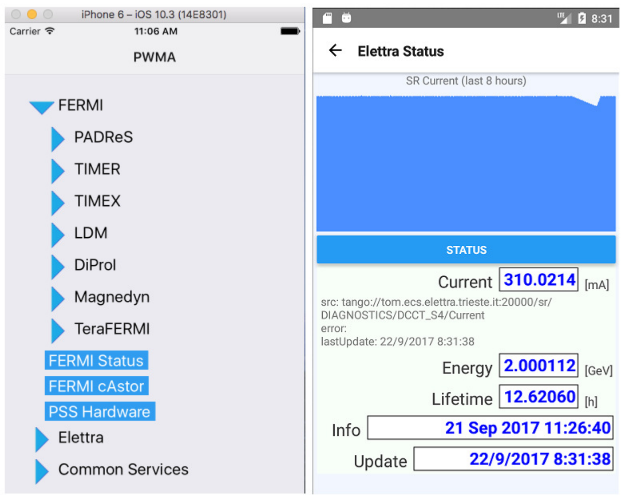
Components are the bricks of PWMA architecture. Almost every component was implemented in two versions: one for the web (using React) and the other for mobile (React Native). Components depends only from React or React Native, they don't depend on WebSockets or on the control system. In order to avoid the duplication of many components and to depend on other parts, our effort was to keep the number of components as little as possible. So our aim was to made the components as generic as possible. The components are:

- **PwmaScalar**, used for any scalar value (i.e. a number or a string) the value can be preceded by a label or followed by a unit. User can see extra data tapping or leaving the mouse pointer on the value
- **PwmaEncodedArray**, a set of values can be encoded as integer or boolean
- **PwmaLedArray**, a set of boolean values displayed through red and green leds
- **PwmaChart**, shows an historical chart of a scalar numeric value
- **PwmaContiner**, a group of components with a title that can be hidden
- **PwmaCredentials**, a modal for entering user's credentials
- **PwmaInput**, for entering data, either a button or some scalar value
- **PwmaMain**, a frame for all other components
- **PwmaModal**, opens a modal (used also by PwmaCredentials)
- **PwmaStorage**, save data in local storage
- **PwmaTree**, show a series of options organized as a tree
- **PwmaLabel**, a text with a style

Screens

Screens are made of components like rooms are made of bricks. Normally they are made only of PWMA components, because if a PWMA component behaviour is independent on DOM or React Native components; this allows the same screen to work both on web and mobile version.

There are two kind of screens: dynamically loaded and static (or custom). Dynamically loaded screens are downloaded from a web server at runtime and instantly rendered. The advantage is to choose from a potentially unlimited list of screens and the user can easily make custom screens. Dynamic screens are built using a designer (available only on PwmaWeb). Variables can be selected from some external GUIs and dropped onto the designer. Also components can be dragged and dropped inside the designer thanks to a specific module [26]. Dynamically loaded screens can do only a limited set of operations and display patterns. Static screens are compiled and embedded in the app (Fig. 2). Their purpose is to implement more complex tasks. A complex task may be split in more then one screen. The static screens already implemented are cAstor and machine status (which are similar to the elettrApp version), a new screen was implemented for the PSS (Personal Safety System), from this screen user can reach about 70 dynamic screens, each monitoring the passivation of a few input/output channels.



Current 310.0214 [mA]
src: tango://tom.ecs.elettra.trieste.it:20000/sr/DIAGNOSTICS/DCCT_S4/Current
error:
lastUpdate: 22/9/2017 8:31:38
Energy 2.000112 [GeV]
Lifetime 12.62060 [h]
Info 21 Sep 2017 11:26:40
Update 22/9/2017 8:31:38

Figure 2: screenshot of the starter, shot from iOS emulator (Xcode) on the left and screenshot of a custom screen (Elettra Status), shot from Android Emulator on the right.

Starter

The starter (Fig. 2) allows to reach the main screen of all tasks and provides the tools for connecting to the PWMA server through a web socket. Each task is launched with an optional parameter; this parameter is a string that can be a JSON encoded complex type reduced to a string. The PwmaMobile starter utilize the react-navigation module, which substituted the previous navigation tools in late spring 2017; we plan to use it also for the PwmaWeb starter.

CONCLUSION

We implemented most of the ideas discussed on the Tango mailing list on May 2015 [27] and even some from a paper published in 2007 [28]. We tested some different web/app technologies. Our concerns weren't only efficiency, scalability and learning curve, but also independence from specific technologies; we recognize that we still have some heavy dependencies, but we constantly try to minimize them. Particularly in the JavaScript community, there are so many libraries, tools, loaders, etc. and almost each of them are constantly and quickly developing so that the expression "JavaScript fatigue" [20] was invented.

Another effort has been taken to grant the user a wide range of possible customizations. New screens and alarms can be created easily. We plan to add many more customization options (e.g. formulae).

We are evaluating the possibility to connect other control systems or to interface to completely new fields such as augmented reality. This technology can allow to detect physical devices and give to operators informations about the state of devices and procedures to be operated locally (e.g. how to reset a power supply).

ACKNOWLEDGEMENT

We would like to thank Giulio Gaio for some smart ideas.

REFERENCES

[1] Web Application, https://en.wikipedia.org/wiki/Web_application
[2] R. K. Camden, *Apache Cordova in Action*. Greenwich, CT, USA: Manning Publications, 2015.
[3] a proof of concept of dynamic templates can be found at https://github.com/luciozambon/ProgressiveHybridApp
[4] JQuery, http://jquery.com
[5] Vanilla JavaScript, https://stackoverflow.com/questions/20435653/what-is-vanillajs
[6] Bootstrap, http://getbootstrap.com
[7] Mobile first, https://en.wikipedia.org/wiki/Responsive_web_design
[8] Ionic, http://ionicframework.com
[9] AngularJS, https://angularjs.org
[10] eGiga2m, https://github.com/luciozambon/eGiga2m
[11] Astor, http://www.esrf.eu/computing/cs/tango/tango_doc/tools_doc/astor_doc/index.html
[12] SSE, https://html.spec.whatwg.org/multipage/server-sent-events.html
[13] CQRS, https://en.wikipedia.org/wiki/Command%E2%80%93query_separation
[14] WebSocket, https://www.w3.org/TR/websockets
[15] AngularJS and web components, https://pascalprecht.github.io/2014/10/25/integrating-web-components-with-angularjs
[16] React, https://facebook.github.io/react
[17] React Native, https://facebook.github.io/react-native
[18] A. Boduch, *React and React Native*. Birmingham, UK: Packt Publishing, 2017.

[19] E. Masiello, J. Friedmann, *Mastering React Native*. Birmingham, UK: Packt Publishing, 2017

[20] M. Bertoli, *React Design Patterns and Best Practices*. Birmingham, UK: Packt Publishing, 2017

[21] AirBnB lint configurations `https://www.npmjs.com/package/eslint-config-airbnb`

[22] JSON, `http://www.json.org/`

[23] FCM, `https://firebase.google.com/docs/cloud-messaging/`

[24] PyEpics, `https://github.com/pyepics/pyepics`

[25] Python WebSocket server, `https://github.com/Pithikos/python-websocket-server`

[26] React DnD, `https://react-dnd.github.io/react-dnd/`

[27] Message in Tango mailing list, `https://lists.tango-controls.org/wws/arc/info/2015-05/msg00048.html`

[28] M. Pelko, K. Zagar, L. Zambon, and A. J. Green, “Canone - A Highly-Interactive Web-Based Control System Interface”, in *Proc. ICALEPCS’07*, Knoxville, TN, USA, Oct. 2007, paper TPPA20, pp. 129-131.